Technische Universität Braunschweig

Institut für Regelungstechnik

Fusion von Umfeld wahrnehmenden Sensoren in städtischer Umgebung

6. Juni 2014

Sebastian Ohl

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik der Technischen Universität Carolo-Wilhelmina zu Braunschweig zur Erlangung des Grades eines Doktors der Ingenieurwissenschaften (Dr.-Ing.) genehmigte Dissertation von Sebastian Ohl aus Braunschweig.

Eingereicht am:	9. Februar 2014	Vorsitzender:	Prof. Dr.-Ing. Thomas Form
Mündliche Prüfung:	6. Juni 2014	1. Referent:	Prof. Dr.-Ing. Markus Maurer
Druckjahr:	2014	2. Referentin:	Prof. Dr.-Ing. Ina Schaefer

Fusion von Umfeld wahrnehmenden Sensoren in städtischer Umgebung

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines Doktors
der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von Sebastian Ohl
aus Braunschweig

Eingereicht am:	9. Februar 2014
Mündliche Prüfung am:	6. Juni 2014
1. Referent:	Prof. Dr.-Ing. Markus Maurer
2. Referentin:	Prof. Dr.-Ing. Ina Schaefer
Druckjahr:	2014

Dissertation an der Technischen Universität Braunschweig,
Fakultät für Elektrotechnik, Informationstechnik, Physik

Vorwort

Die vorliegende Arbeit entstand in meiner Zeit am Institut für Regelungstechnik vom September 2007 bis zum Juli 2012. In dieser Zeit hatte ich die Möglichkeit mich intensiv mit dem Gebiet des automatischen Fahrens und der zugehörigen Umfeldwahrnehmung auseinanderzusetzen.

Zu Beginn meiner Arbeit als wissenschaftlicher Mitarbeiter konnte ich an den Finalwettkämpfen der DARPA Urban Challenge im Team CarOLO der TU Braunschweig teilnehmen. Diese Arbeit basiert nicht zuletzt auf den Erfahrungen, die ich während dieser Zeit machte. Im Frühjahr 2008, als Prof. Dr.-Ing. Markus Maurer den Lehrstuhl für elektronische Fahrzeugsysteme übernahm, wurde der Grundstock für diese Arbeit mit der Definition des Projekts Stadtpilot als Folgeprojekts des DARPA Urban Challenge Teams CarOLO gelegt. In den folgenden Jahren entwickelte ich zusammen mit vielen Studierenden und Kollegen das automatische Fahrzeug Leonie, das die Anwendung für das hier beschriebene Umfelderfassungssystem darstellt.

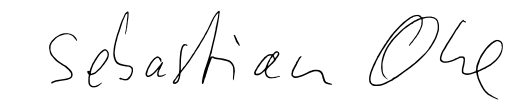
Mein besonderer Dank gilt meinen Teammitgliedern, die mich in den letzten Jahren begleitet haben. Durch die Zusammenarbeit unterschiedlicher Lehrstühle und Fachrichtungen wurde uns allen immer wieder bewusst, dass jeder doch eine etwas andere Sichtweise auf die Dinge hat. Trotz manchmal intensiver Diskussionen wurde, das gemeinsame Ziel vor Augen, jedoch immer wieder ein Kompromiss zur Lösung der aktuellen Herausforderung gefunden.

Ferner möchte ich meinen Korrektorinnen und Korrektoren Asem Eltaher, Peter Bergmiller, Felix Klose, Bernd Lichte, Annemarie Ohl, Rainer Ohl, Prof. Dr.-Ing. Ina Schaefer, Jens-Wolfhard Schicke-Uffmann, Ole Schütt und Simon Ulbrich für die konstruktive Kritik und die mit der Korrektur verbundene Arbeit danken.

Darüber hinaus danke ich Prof. Dr.-Ing. Markus Maurer für die konstruktive Kritik meiner wissenschaftlichen Beiträge. Ohne diese Hinweise und Verbesserungsvorschläge hätte diese Arbeit nicht die nun erreichte wissenschaftliche Qualität erlangt.

Während meiner Arbeit wurden einige Aufgaben von Studenten in Form von Abschlussarbeiten oder Tätigkeiten als wissenschaftliche Hilfskraft übernommen. Nicht alle Arbeiten konnten in dieser Dissertation verwendet werden, einiges davon ist jedoch eingeflossen. In diesem Sinne danke ich den von mir betreuten Studenten Yang Gao, Lars Kählke, Schanchun Liu, Gregor Marek, Matthias Müller, Thore Schütt, Bruno Favoreto Soares, Thorsten Volz und Jan Timo Wendler für ihre Mitarbeit.

Abschließend bleibt mir noch, meinen Kollegen am Lehrstuhl zu danken. Mir hat die Arbeit und die Zusammenarbeit mit ihnen am Institut für Regelungstechnik viel Spaß bereitet. So blicke ich mit einem weinenden und einem lachenden Auge in die Zukunft und schließe dieses Kapitel meines Lebens ab.



Braunschweig, 6. Juni 2014

II

Kurzfassung: Fusion von Umfeld wahrnehmenden Sensoren in städtischer Umgebung

Dipl.-Inform. Sebastian Ohl

In den letzten Jahren haben sich moderne innovative Fahrerassistenzsysteme (z. B. Night-Vision oder Aktivlenkung) immer mehr zu Alleinstellungsmerkmalen und Kaufargumenten von Automobilen entwickelt (BMW Group München, 2009, Seite VI). Zunächst wurden in den 1990er Jahren Systeme wie ABS, ASR und ESC, die sich auf fahrdynamische Eigenschaften konzentrierten, eingeführt. Inzwischen sind sie als Serienausstattung in vielen Fahrzeugen zu finden und seit November 2011 verpflichtend für neue Fahrzeugtypen in der Europäischen Union (Europäisches Parlament, 2009). Mitte der 1990er Jahre waren die ersten komplexeren Systeme in Fahrzeugen der Oberklasse zu finden. Diese nahmen den Längsverkehr im Fahrzeugumfeld wahr, um ihre Aufgabe erfüllen zu können. Im Jahr 1995 wurde beispielsweise von Mitsubishi im Modell Diamante ein adaptiver Tempomat eingeführt, der mithilfe eines Lasersensors den Abstand und die Geschwindigkeit der vorausfahrenden Fahrzeuge maß und die eigene Geschwindigkeit entsprechend anpasste (Watanabe u. a. (1995) zitiert nach Winner u. a. (2009, Seite 479)). Moderne Fahrerassistenzsysteme wie Notbremsassistent, ACC-Stop&Go oder Einparkassistent bis hin zur vollkommenen Entlastung des Fahrers durch eine vollständig automatische Fahrzeugführung sind ohne Sensoren zur Umfeldwahrnehmung nicht vorstellbar.

In der Vergangenheit wurden Fahrerassistenzsysteme und die notwendigen Sensoren oft als Paket entwickelt. Dies führte dazu, dass bei der Ausrüstung eines Fahrzeugs mit mehreren Systemen keine Synergien zwischen den Sensorsystemen genutzt werden konnten. Denkbar wäre hier beispielsweise, dass ein Notbremsassistent die Kamera einer Fahrstreifenwahrnehmung nutzt, um seine Daten zu verifizieren, ohne dass eine weitere Kamera ins Fahrzeug eingerüstet werden muss. Diese Zentrierung des Informationsflusses, und so die Nutzung eines Sensors durch mehrere Assistenzfunktionen, kann mithilfe eines zentralen Umfeldwahrnehmungssystems geschehen. Es wird mit allen im Fahrzeug verbauten Sensoren verbunden und stellt anschließend allen Assistenzsystemen im Fahrzeug die aufbereiteten und miteinander fusionierten Daten zur Verfügung.

Um eine effektive Entwicklung von Umfeldwahrnehmungssystemen zu gewährleisten, sollten große Teile von vorherigen Entwicklungen einfach wiederverwendet und neue sowie bereits in früheren Projekten genutzte Sensoren ohne großen Aufwand an ein System angebunden werden können. Die Eigenschaften des Systems definieren sich dabei durch die Anforderungen der jeweiligen Assistenzfunktionen.

Im Rahmen dieser Arbeit wurde eine Softwarearchitektur zur Entwicklung von Umfeldwahrnehmungssystemen entwickelt (Ohl u. Maurer, 2011b; Ohl u. a., 2011). Basierend auf einer allgemeinen Beschreibung der Arbeitsdomäne und einer Kontextdefinition werden Anforderungen an die Softwarearchitektur aufgestellt und anschließend die statische sowie die dynamische Architektursicht behandelt.

Die Softwarearchitektur teilt sich in drei Ebenen auf. Die Sensorebene beschreibt die Dekodierung von sensorspezifischen Protokollen und die Konvertierung in eine einheitliche Containerdatenstruktur. Darauf aufbauend schließt sich die Fusionsebene an. Als Sukzessor der Sensorebene konsumiert sie die Containerdatenstruktur und verarbeitet sie in einer gitter- bzw. objekthypothesenbasierten Sensordatenfusion. Als dritte Ebene wird die Applikationsebene eingeführt. Sie dient als Datensenke für die Fusionsebene und enthält die verarbeitenden Assistenzsysteme.

Aus Effizienzgründen werden innerhalb der Fusionsebene für die gitter- und objekthypothesenbasierte Fusion je getrennte Architekturen aufgestellt. Diese lassen sich jedoch auf die Architektur der Arbeitsdomäne zurückführen. Innerhalb der einzelnen Architekturen wird auf ein Höchstmaß an Unabhängigkeit zwischen den Verarbeitungsstufen sowie auf eindeutig spezifizierte Schnittstellen geachtet. Dies ermöglicht eine Wiederverwendung von Algorithmen und Komponenten in späteren Projekten und verkürzt so die Entwicklungszeit.

An die Beschreibung der Softwarearchitektur schließt sich deren Realisierung an. Sie wurde im Rahmen des Projekts „Stadtpilot“ durchgeführt. Das Projekt „Stadtpilot“ widmet sich dem teilautomatischen Fahren im innerstädtischen Bereich (Nothdurft u. a., 2011b). Als Projektziel wird hierbei die automatische Umrundung des Stadtkerns von Braunschweig im öffentlichen Straßenverkehr angestrebt. Um dies zu erreichen, wurde der Versuchsträger mit Radar- und Lasersensoren, einer Trägheitsplattform, einem Datenverarbeitungssystem sowie einer elektronischen Ansteuerung der Fahrzeugaktorik ausgerüstet.

Zur Verarbeitung der Umfeld wahrnehmenden Sensoren des Versuchsträgers wurde sowohl ein gitter- als auch ein objekthypothesenbasiertes Fusionssystem auf Basis der Softwarearchitektur entwickelt. Diese werden je für unterschiedliche Aufgaben eingesetzt. Das objekthypothesenbasierte System teilt sich dabei in eine Hauptfusion sowie eine Vorfusion auf. Die Hauptfusion enthält ein konturschätzendes Kalman Filter (Ohl u. Maurer, 2011a). Dieses ist in der Lage, sein Objekthypothesenmodell den von den Sensoren wahrgenommen Messungen anzupassen. Auf diese Weise beschreibt das Filter unterschiedliche Objekthypothesen mit der jeweils minimalen Anzahl an Stützpunkten. Um eine verlässliche Bestimmung des Konturtyps zu ermöglichen, wird mithilfe einer Ähnlichkeitsfunktion und eines Dempster-Shafer-Filters eine Stabilisierung der Konturschätzung durchgeführt. Ergänzt wird die Hauptfusion durch eine Fusion zur Stabilitätsanalyse von Objekthypothesen. Diese zweite Fusion hat zum Ziel, Objekthypothesen, die sich nicht modellkonform verhalten, zu klassifizieren und auszusortieren. Darüber hinaus werden fehlende Messdaten der Sensoren (z. B. Geschwindigkeitsvektor) aus einem Modell abgeleitet und so verbesserte Initialwerte für die Hauptfusion bereitgestellt. Zusätzlich werden redundante Erfassungsbereiche der Sensoren ausgenutzt, um Fehldetektionen von Objekten zu reduzieren. Dabei wird ein System von logischen Ausdrücken und Bereichen um den Versuchsträger verwendet, um zu entscheiden, ob und wann eine Objekthypothese von der Vorfusion in die Hauptfusion übergeht.

Die gitterbasierte Fusion enthält ein binäres Bayes-Filter. Die Daten dieser Fusion werden

durch drei Verfahren mit unterschiedlichen Zielsetzungen ausgewertet. Das erste Verfahren bestimmt Bereiche, die von den Sensoren nicht vermessen werden können. Diese Daten werden genutzt, um bei Abbiegemanövern Informationen über Verdeckungen zu gewinnen. Das zweite Verfahren nimmt Verengungen im Fahrstreifen aufgrund von abgestellten Fahrzeugen war, wie sie im innerstädtischen Bereich oft durch abgestellte Fahrzeuge vorkommen. Auf Grundlage des von Weiss u. a. vorgestellten Algorithmus lassen sich diese leichten Verengungen detektieren und so eine Verschiebung der Trajektorie veranlassen (Weiss u. a., 2007; Weiss, 2011). Die hier eingesetzte gitterbasierte Fusion ermöglicht vor allem eine akkurate Abbildung von sich nicht bewegenden Objekten. Aus diesem Grund lassen sich mit dem dritten Verfahren aus diesen Daten Objekthypothesen statischer Objekte gewinnen, die wiederum von der objekthypothesenbasierten Fusion verarbeitet werden können. Hierzu wird ein neu entwickelter Algorithmus der Familie der Split&Merge-Algorithmen aus dem Bereich der Robotik eingesetzt. Ihr Hauptanwendungsbereich findet sich in der Auswertung von 1D-Laserscannern. Aufgrund dieser Ausrichtung können Split&Merge-Algorithmen nicht direkt auf Gitterdatenstrukturen angewendet werden. Deshalb wird im Rahmen dieser Arbeit ein 2D-Split&Merge-Algorithmus vorgestellt, der auf der Basis von Gitterdaten Objekthypothesen bildet.

Ergänzt wird die Arbeit durch eine Bewertung der aufgestellten Anforderungen und vorgestellten Algorithmen. Hierbei werden sowohl simulierte Daten, Daten von Messfahrten auf einem Testgelände als auch Aufzeichnungen aus dem öffentlichen Straßenverkehr genutzt.

Stichworte:

- Umfeldwahrnehmung
- Urbanes Umfeld
- Sensordatenfusion
- Erweitertes Kalman-Filter
- Konturtracking
- Softwarearchitektur

Abstract:
Environmental perception using Multi Sensor Data Fusion in Urban Environments

Dipl.-Inform. Sebastian Ohl

Recently, the development of modern innovative driver assistance systems, like night-vision or active steering, gained an emerging interest in the automotive industry (BMW Group München, 2009, p. VI). In an effort to improve vehicle dynamics, in the beginning of the 1990s, systems like ABS, ASC, and ESC, are developed. In the meantime, recently developed vehicles are equipped with these systems, which have become a obligatory requirement in the European Union since November 2011 (Europäisches Parlament, 2009). In the mid-1990s, the first advanced driver assistance system could be found in upper class vehicles. These systems perceive the vehicle's environment to fulfill their task. For example, in 1995 Mitsubishi introduced an adaptive cruise control system in its model Diamante. The core idea is to measure the distance to the preceding vehicle by a laser sensor and adapt the vehicle's velocity accordingly (according to Watanabe u. a. (1995) as cited in Winner u. a. (2009, p. 479)). Modern driver assistance systems, e.g. emergency breaking, ACC-Stop&Go, parking assistance, up to a fully automatic driving system are of no practical use without the aid of environmental perception sensors.

In the past, driver assistance systems and their necessary sensors are developed together as a sole integrated packet. As a result, the usage of synergies between sensors is practically not possible. This consequently leads to further complication like the non-ability to employ an emergency breaking system that uses the camera of a lane detection system to verify its targets without the need to install a second camera in the vehicle. Having the information centrally available would render the usage of one sensor by several driver assistance systems possible. Briefly, the idea behind this is to develop a central perception system that is connected to every sensor to make the fused and aligned data available to other driver assistance systems.

To ensure a effective driver assistance system development, the majority of the previously developed systems should be reused. In addition, sensors from previous projects should be integrated into new systems without a great effort. In this context, it is worthwhile to mention that the characteristics of a new system are defined by the requirements of the particular assistance functions.

In this PhD thesis, a software architecture for defining environmental perception systems is described (Ohl u. Maurer, 2011b; Ohl u. a., 2011). Based on the description of the architecture's domain and its context, requirements are defined. Afterward, the static and the dynamic views of the software architecture are discussed.

The software architecture is divided into three layers. The sensor layer describes the decoding of sensor specific protocols and the conversion of sensor data into a common container structure. The next layer is the fusion layer that, as a successor to the sensor layer, consumes the container data structure. Afterward, this data is processed in a grid-based or in a object hypotheses-based sensor data fusion. As a third layer, the application layer is introduced. This layer works as a data sink for the fusion layer and contains the driver assistance systems.

For efficiency reasons, there are different architectures for the grid-based and the object hypotheses-based sensor data fusions within the fusion layer. Both architectures can be reduced to the domain's architecture. Within the scope of each architecture individually, the processing algorithms and interfaces are defined independently and clearly. As a natural result, this allows the reuse of algorithms and components in other projects and, therefore, reduces the developmental time and cost.

Next, the realization of the above-described architecture is addressed. The architecture is realized within the project "Stadtpilot". The Stadtpilot-project is dedicated to semi-autonomous driving in urban environments (Nothdurft u. a., 2011b). The project's goal is to drive automatically on the inner-city ring of the city Braunschweig among public traffic. To accomplish this task, the test vehicle is equipped with radar and laser sensors, an inertial measurement unit, a computer system as well as a electronic vehicle control system.

The processing of the vehicle's environmental perception systems consists of a grid-based as well as a object hypotheses-based sensor data fusion system. These fusion systems are used simultaneously for different tasks. The main object hypotheses-based data fusion contains a contour classifying Kalman-filter (Ohl u. Maurer, 2011a). This filter can adapt its object hypothesis model to the data of the perception sensors. This way, the filter describes different object hypotheses with the minimal necessary contour point count. To achieve a reliable contour classification, a similarity metric and a Dempster-Shafer-filter is used to stabilize the contour classification. The second data fusion detects object hypotheses, which do not comply with a certain object hypothesis model. Furthermore, missing measurement values (e. g. velocity vector) are derived from a physical model to improve the initial tracking of the main fusion. In addition, redundant detection areas of the perception sensors are used to reduce measurement errors and ghost object hypotheses. Thereby, a system of logical expression is used to decide whether and when an object hypothesis moves from the preceding to the main fusion system.

The grid-based data fusion contains a binary Bayes-filter. Three different algorithms with different goals process the grid-based fusion's data. The first algorithm detects areas that can currently not be measured by the sensors. For example, this data can be used during turn maneuvers. The second algorithm sense lane constrictions by parking vehicles, as they occur in inner-city environments very often. Based on the algorithm of Weiss u. a., these constrictions can be detected easily and the test vehicle's trajectory can, therefore, be accordingly adapted (Weiss u. a., 2007; Weiss, 2011). Due to its configuration, the grid-based sensor data fusion of project Stadtpilot represents especially static objects. Therefore, the third algorithm is developed to create object hypotheses of static objects from this data. Afterward, these object hypotheses could be sent to the object hypotheses-based fusion system. The newly developed algorithm is based on the split&merge-algorithm family from the robotics domain. In this domain, the algorithms are mainly used to process 1D-

laserscanner data. Consequently, these algorithms cannot be deployed to 2D-grid-based data structures. Therefore, a 2D-split&merge-algorithm is developed. It can process grid-based data structures and create object hypotheses of the represented objects.

In addition to the research work done in this thesis, it is enriched by an assessment of the defined requirements and presented algorithms. To do so, simulated data as well as real measured data from a closed test track and Braunschweig's inner-city ring are used.

Keywords:

- Environmental perception
- Urban environment
- Sensor data fusion
- Extend Kalman-filter
- Contour tracking
- Software architecture

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programmabschnitte

Symbolverzeichnis

$\hat{x}[k]$	Systemzustandsvektor
$F[k]$	Systemmatrix
$\tilde{y}[k]$	Innovationsvektor
$\breve{P}$	Modellübergangswahrscheinlichkeit eines IMM-Filters
$\overline{T}$	Menge von Tracks
$\overline{S}$	Menge von Messdaten
M_m^t	Massenverteilung des Tracks bei der Schätzung des Bewegungsmodells
M_m^m	Massenverteilung des Messwerts bei der Schätzung des Bewegungsmodells
2_m^Θ	Wahrscheinlichkeitspotenzraum der Schätzung des Bewegungsmodells
$2_{Polyline}^\Theta$	Wahrscheinlichkeitspotenzraum der Schätzung der Kontur (Sensor mit Polygon-Modell)
$2_{Strecke}^\Theta$	Wahrscheinlichkeitspotenzraum der Schätzung der Kontur (Sensor mit Strecken-Modell)
2_{Punkt}^Θ	Wahrscheinlichkeitspotenzraum der Schätzung der Kontur (Sensor mit Punkt-Modell)
$\Theta_A(s)$	Turning-Funktion nach Arkin u. a. (1991)
D_B^A	Ähnlichkeitsfunktion von zwei Konturen
ε	Schwellwert für den jeweiligen Algorithmus (z. B. Gateingschwellwert)
p_{free}	Wert für die Wahrscheinlichkeit der Belegung einer Zelle
$p_{\overline{free}}$	Wert für die Wahrscheinlichkeit der Nicht-Belegung einer Zelle
Ξ	Wert für die Beobachtbarkeit einer Gitterdatenstruktur
μ_i	Wahrscheinlichkeit für Modell i eines IMM-Filters
$E(\hat{x})$	MSE des Systemzustandsvektors $\hat{x}$ unter Nutzung des Referenzvektors x

Glossar

ACC	Adaptive Geschwindigkeitsregelanlage (engl. Adaptive Cruise Control)
CAN-Bus	Datenbus (Schumny u. Ohl, 1994, Seite 288)
CA	Constant Acceleration
CT	Coordinated Turn
CV	Constant Velocity
DDS	Data Distribution System der Object Management Group
DGPS	Differential Global Positioning System
Echtzeitfähig	Eine Datensenke verarbeitet die Eingangsdaten schneller, als die Datenquelle sie bereitstellt
Ego-Fahrzeug	Eigener Versuchsträger
Ethernet-Netzwerk	Datenbus (Schumny u. Ohl, 1994, Seite 318)
ESC	Elektronische Stabilitätsprogramm (engl. Electronic Stability Control)
Fahrbahn	Bereich einer Straße von Randstein zu Randstein
Fahrspur	Zurückgelegter Weg eines Fahrzeugs
Fahrstreifen	Durch eine Markierung festgelegter Fahrbereich auf einer Fahrbahn
Fahrzeugkoordinaten	Relativ zum Ego-Fahrzeug definierte Position
Fusionsmodul	Abgeschlossene Einheit zur Fusion von Daten Umfeld wahrnehmender Sensoren (siehe Abschnitt 9)
Fusionszyklus	Abfolge der in Abschnitt 8 beschriebenen Basisfunktionen
Gateing	Schwellwertentscheidung
Gitterkoordinaten	Diskretisierte Position relativ zum Ursprung einer Instanz der *Layer*-Komponente
Gitterzelle	Kleinstes Element einer Gitterdatenstruktur
IEPF	Iterative End-Point-Fit
IMM-Filter	Interacting-Multiple-Model-Filter
Kontur	Geordnete Liste von Punkten im 2D-Raum, welche durch Strecken verbunden sind
Konturpunkt	Einzelner Stützpunkt einer Kontur
Layer	2D-Gitterdatenstruktur
LKW	Lastkraftwagen
MMS	Mensch-Maschine-Schnittstelle
MSE	Mean Squared Error
Objekthypothese	Annahme über die Existenz eines Objekts in der realen Umwelt, dass durch eine Menge von Zustandsparametern beschrieben wird

Objekt	Physisch existierendes Element der realen Umwelt
PKW	Personenkraftwagen
Relevantes Objekt	Hindernis für das Ego-Fahrzeug oder anderer Verkehrsteilnehmer
SDF	Sensordatenfusion
Sensorkoordinaten	Position relativ zum Ursprung des Sensors
Track	Bekannte Objekthypothese der Sensordatenfusion
Tracking	Verfolgung einer oder mehrerer Objekthypothesen durch eine Sensordatenfusion
Trajektorie	Geplanter Fahrweg
TTC	Time to Collision
UML	Unified Modeling Language
Verkehrsteilnehmer	Fußgänger, Zweiradfahrer, PKW, LKW
Weltkoordinaten	Ortsfeste Position in der Welt

Nomenklatur	Beschreibung Automatisierungsgrad und Erwartung des Fahrers
Driver Only	Fahrer führt dauerhaft (während der gesamten Fahrt) die Längsführung (Beschleunigen / Verzögern) und die Querführung (Lenken) aus.
Assistiert	Fahrer führt dauerhaft entweder die Quer- oder die Längsführung aus. Die jeweils andere Fahraufgabe wird in gewissen Grenzen vom System ausgeführt. ■ Der Fahrer muss das System dauerhaft überwachen. ■ Der Fahrer muss jederzeit zur vollständigen Übernahme der Fahrzeugführung bereit sein.
Teilautomatisiert	Das System übernimmt Quer- und Längsführung (für einen gewissen Zeitraum und/oder in spezifischen Situationen). ■ Der Fahrer muss das System dauerhaft überwachen. ■ Der Fahrer muss jederzeit zur vollständigen Übernahme der Fahrzeugführung bereit sein.
Hochautomatisiert	Das System übernimmt Quer- und Längsführung für einen gewissen Zeitraum in spezifischen Situationen. ■ Der Fahrer muss das System dabei nicht dauerhaft überwachen. ■ Bei Bedarf wird der Fahrer zur Übernahme der Fahraufgabe mit ausreichender Zeitreserve aufgefordert. ■ Systemgrenzen werden alle vom System erkannt. Das System ist nicht in der Lage, aus jeder Ausgangssituation den risikominimalen Zustand herbeizuführen.
Vollautomatisiert	Das System übernimmt Quer- und Längsführung vollständig in einem definierten Anwendungsfall. ■ Der Fahrer muss das System dabei nicht überwachen. ■ Vor dem Verlassen des Anwendungsfalles fordert das System den Fahrer mit ausreichender Zeitreserve zur Übernahme der Fahraufgabe auf. ■ Erfolgt dies nicht, wird in den risikominimalen Systemzustand zurückgeführt. ■ Systemgrenzen werden alle vom System erkannt, das System ist in allen Situationen in der Lage, in den risikominimalen Systemzustand zurückzuführen.

Tabelle 0.4: Definition verschiedener Automatisierungsgrade nach Bundesanstalt für Straßenwesen (2012, Seite 9)

Teil I: Einführung

Innerhalb des Teils I wird zunächst eine kurze Einführung in das Themengebiet der vorliegenden Arbeit gegeben. Daran schließt sich ein Überblick über aktuelle Projekte im Bereich des automatischen Fahrens an. Darauf folgend wird auf Sensortechnologien und Filterverfahren zur Umfeldwahrnehmung eingegangen. Eine Zusammenfassung von Architekturen zur Umfeldwahrnehmung schließt den Stand der Technik ab. Vervollständigt wird Teil I mit einer Beschreibung des Projekts, in dem die beschriebene Architektur zur Umfeldwahrnehmung umgesetzt wird.

1 Einleitung

Mobilität ist in der heutigen Gesellschaft Teil des Lebensgefühls. Wo früher scheinbar unüberwindbare Entfernungen Hindernisse darstellten, nutzen wir heute zur Überbrückung elektronische Kommunikationsmittel, das Automobil oder Flugzeuge. So waren z. B. im Jahr 2011 ca. 50,9 Mio. Kraftfahrzeuge in Deutschland zugelassen (Statistisches Bundesamt, 2012, Seite 104). Der Einsatz dieser Fahrzeuge ist trotz vieler Sicherheitsmaßnahmen mit einem gewissen Risiko behaftet und führte im Jahr 2011 zu 2,3 Mio. polizeilich erfassten Verkehrsunfällen in Deutschland, von denen 4002 mit tödlichem Personenschaden ausgingen (Statistisches Bundesamt, 2012, Seite 103).

Betrachtet man die Entwicklung von tödlichen Personenschäden, so ist eine deutliche Reduktion bis zum Jahr 2010 zu verzeichnen. Dieser Rückgang ist u. a. auf die erhebliche Ausweitung der Sicherheitsfunktionen in modernen Kraftfahrzeugen zurückzuführen. Aktuell homologierte Fahrzeuge verfügen über eine Vielzahl von Airbags, Versteifungen der Karosserie oder moderne Fahrerassistenzsysteme.

Neben den passiven Sicherheitsfunktionen rückt in den letzten Jahren die aktive Sicherheit immer mehr in den Fokus von Herstellern und Käufern. Als erstes System brachte Mercedes Benz 1995 das elektronische Stabilitätsprogramm (ESC) in der S-Klasse in Serie. Dabei wird durch die Messung der Gierrate eine bevorstehende nicht beherrschbare Fahrsituation wahrgenommen und durch das Abbremsen einzelner Räder die Kontrolle über das ausbrechende Fahrzeug wiedererlangt. Das ESC hat sich inzwischen als unverzichtbares Fahrerassistenzsystem erwiesen und wird von der Europäischen Union seit November 2011 für alle neu homologierten Fahrzeuge verpflichtend gefordert (Europäisches Parlament, 2009).

Das ESC beobachtet zur Erfüllung seiner Aufgabe ausschließlich das eigene Fahrzeug. Durch den Einsatz von Umfeld wahrnehmenden Sensoren (z. B. Radar-, Lidar- oder Kamerasensor) wird es möglich, auch auf andere Verkehrsteilnehmer zu reagieren. So führte Mitsubishi bereits 1995 ein System ein, das den Abstand zum vorausfahrenden Fahrzeug durch einen Lasersensor maß und die Geschwindigkeit entsprechend anpasste (Watanabe u. a. (1995) zitiert nach Winner u. a. (2009, Seite 479)). Darauf folgend wurden in den letzten Jahren Fahrerassistenzsysteme mit maschineller Wahrnehmung in immer neuen Ausprägungen und Varianten auf den Markt gebracht und so ein Beitrag zur Verbesserung der Fahrzeugsicherheit oder des Fahrkomforts geleistet.

Den meisten aktuellen Assistenzsystemen ist gemein, dass sie stark von der Wahrnehmungsleistung der genutzten Sensoren abhängen. So kann beispielsweise ein Fahrstreifenhalteassistent dem Fahrstreifen nur folgen, wenn er nicht durch starke Verschmutzung oder schäumende Gischt verdeckt wird. Genauso kann eine adaptive Geschwindigkeitsregelanlage auf ein einscherendes Fahrzeug nur reagieren, wenn es vorher durch einen Sensor wahrgenommen wurde.

1.1 Motivation und wissenschaftlicher Beitrag

Eine Möglichkeit dieser Herausforderung zu begegnen, ist der Einsatz eines Fusionssystems, welches unterschiedliche Sensoren zu einer gemeinsamen Abbildung der Fahrzeugumgebung

zusammenführt (Khaleghi u. a., 2011, Seite 1). Durch den Einsatz von unterschiedlichen Sensoren und Sensortechnologien kann es zu Konflikten zwischen den Daten kommen. Die Auflösung von Unvollkommenheiten, Inkonsistenzen, Zuordnungsproblemen und Vergleichsproblemen zwischen den Messdaten von Sensoren stellt die große Herausforderung bei der Erstellung von Fusionssystemen dar. Andererseits liegt in den Unterschieden der Sensoren die große Chance beim Einsatz dieser Systeme. Durch unterschiedliche Messprinzipien können beispielsweise Schwächen der einen Technologie durch die Stärken einer anderen abgeschwächt oder ein eingeschränkter Erfassungsbereich eines Sensors durch den eines anderen ergänzt werden.

Zur Umsetzung eines Sensordatenfusionssystems können eine Vielzahl von unterschiedlichen Algorithmen eingesetzt werden. Die Auswahl hängt stark von der Zielsetzung des Datenkonsumenten ab. Vergleicht man beispielsweise ein Notbremssystem mit einer adaptiven Geschwindigkeitsregelanlage, so sind die Anforderungen meist grundlegend verschieden. Während eine adaptive Geschwindigkeitsregelanlage Daten mit einer hohen Stabilität akzeptiert, so verlangt ein Notbremssystem vor allem gesicherte Informationen. Andererseits ist für eine adaptive Geschwindigkeitsregelanlage die Latenz bis zur erstmaligen Detektion eines Objekts eher nachrangig, während für ein Notbremssystem gerade diese Größe von erheblicher Bedeutung ist.

Diese einfachen Beispiele zeigen, dass Fusionssysteme optimalerweise an ihre Datensenken angepasst sind. Leider führt diese Tatsache dazu, dass spezialisierte Lösungen entwickelt und weitergepflegt werden müssen. Um dies zu vereinfachen, wurden bereits durch Darms und Munz unterschiedliche Systeme vorgeschlagen. Darms schlägt in seiner Arbeit eine Basis-Systemarchitektur zur Sensordatenfusion vor (Darms, 2007). Diese beschreibt eine Menge von Funktionen, die benötigt werden, um eine Sensordatenfusion zu erstellen. Munz geht in seiner Arbeit einen anderen Weg. Er beschreibt ein Sensordatenfusionssystem, das durch den Einsatz von probabilistischen Beschreibungsformen in der Lage ist, eine größere Bandbreite an Anwendungen zu unterstützen (Munz, 2011).

Im Rahmen dieser Arbeit wird der Ansatz von Darms auf eine andere Weise weiterentwickelt. Mithilfe der vorgestellten Architektur kann ein Fusionssystem aus einzelnen Softwarebausteinen ganz nach den Anforderungen der konsumierenden Applikation erstellt werden. Auf diese Weise können einmal entwickelte Bausteine in Folgeprojekten weiterhin genutzt werden und der Entwicklungsaufwand wird so reduziert. Darüber hinaus beschreiben weder Darms noch Munz eine Architektur zur gitterbasierten Datenfusion.

Angewendet wurde die Architektur im Projekt Stadtpilot der TU Braunschweig für Fahrten auf dem Testgelände des Projekts. In diesem Forschungsvorhaben wird ein Fahrzeug entwickelt, das im teilautomatisierten Betrieb die Herausforderungen des Braunschweiger Stadtrings beherrscht. Insgesamt werden vier unterschiedliche Fusionsmodule umgesetzt: drei objekthypothesenbasierte Fusionsmodule und ein gitterbasiertes Fusionsmodul. Die Realisierung der Architektur basiert auf den Erfahrungen zur Umfelderfassung des Vorgängerprojekts CarOLO (Effertz, 2008; Ohl, 2007). Hierbei wurde die objekthypothesenbasierte Wahrnehmung der Umgebung durch offene Polygonzüge weiterentwickelt und um eine Formschätzung auf der Basis der Evidenztheorie erweitert. Für die gitterbasierte Fusion lag der Schwerpunkt auf der Auswertung der Gitterdatenstruktur. Hier wurde beispielsweise ein Algorithmus zur Objekthypothesenbildung aus Gitterdaten umgesetzt.

1.2 Aufbau der Arbeit

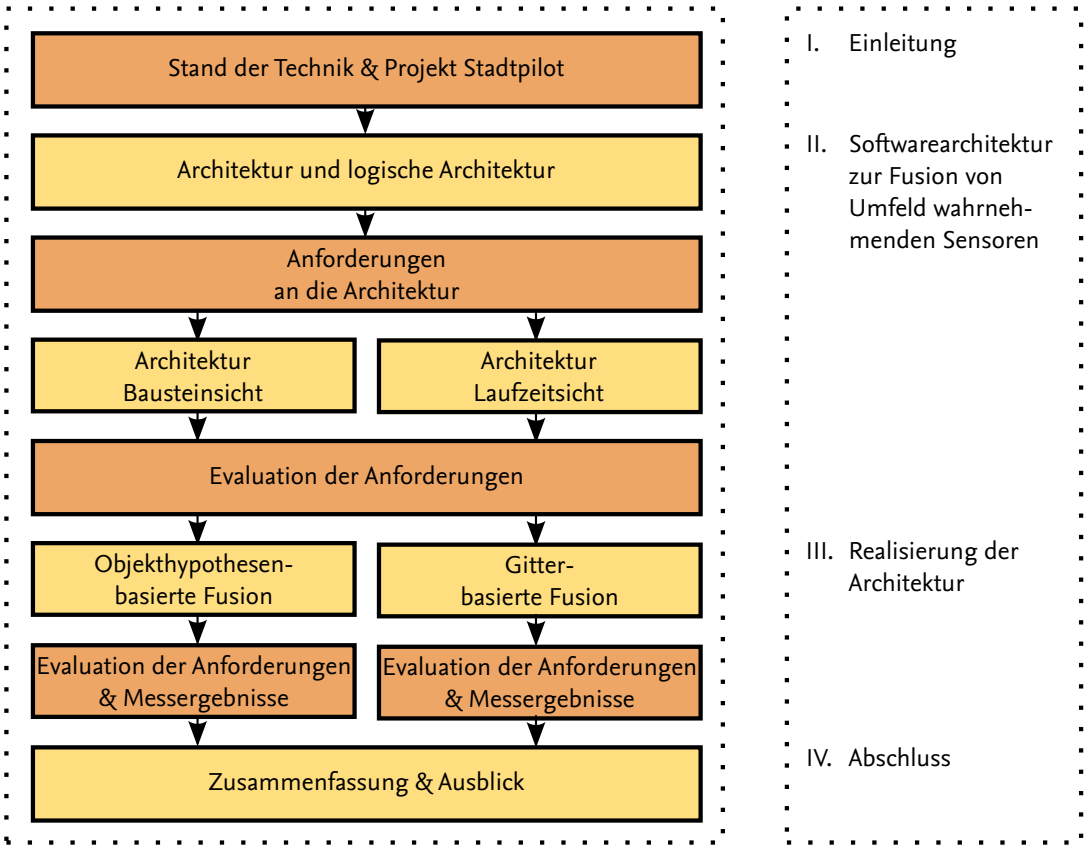


Abbildung 1.1: Aufbau der Arbeit

Der Aufbau der Arbeit ist in Abbildung 1.1 dargestellt. In Teil I wird zunächst in den Stand der Technik zum automatischen Fahren eingeführt. Anschließend wird auf Technologien zur Fusion von Umfeld wahrnehmenden Sensoren eingegangen. Abgeschlossen wird dieser Teil mit einer Übersicht über das Projekt Stadtpilot, welches den Rahmen für die Umsetzung bildet. Teil II beschreibt zunächst die Definition des Architekturkontextes gefolgt von einer logischen Architektur zur Umfeldwahrnehmung im Kraftfahrzeug. Dabei wird das Problem der Fusion auf vier Grundelemente reduziert. Daran schließt sich die Definition der Anforderungen an die Umfeldwahrnehmungsarchitektur an. Anschließend wird die Softwarearchitektur aus Baustein- und Laufzeitsicht dargestellt und diese hinsichtlich der erfüllten Anforderungen evaluiert. Die Realisierung der Architektur findet sich in Teil III. Hier werden zunächst weitere projektspezifische Anforderungen eingeführt, welche anschließend für den Bereich der objekthypothesen- und gitterbasierten Sensordatenfusion umgesetzt werden. Die so beschriebenen Algorithmen werden schließlich mit den aufgestellten Anforderungen abgeglichen. Abgeschlossen wird die Arbeit mit einer Zusammenfassung und einem Ausblick.

2 Stand der Technik

Mit der Forschung zum Thema automatisches Fahren wurde bereits Ende der 1970er Jahre begonnen (Thompson, 1977; Tsugawa u. a., 1979). In Europa folgte dieser Schritt mit dem ersten großen Forschungsprojekt in diesem Bereich im Jahr 1986. Im Forschungsvorhaben PROMETHEUS[1] arbeiteten Forschungseinrichtungen und Fahrzeughersteller aus 19 europäischen Ländern in diversen Einzelprojekten zusammen (Kemeny, 1990). Im Oktober 1994 wurden die Ergebnisse und Prototypen auf der Abschlussveranstaltung in Paris vorgestellt.

Mitte der 1990er Jahre wurden eine Reihe von automatischen Langstreckenfahrten veröffentlicht. So wurde z. B. im November 1995 eine automatische Fahrt von München nach Odense (DK) von der Gruppe um Prof. Dickmanns der Universität der Bundeswehr in München durchgeführt (Maurer u. a., 1996; Maurer, 2000, Seite 135). Die 1758 km lange Strecke wurde dabei zu ca. 95 % automatisch zurückgelegt. Während der Fahrt auf der Autobahn wurde dabei die Längs- und Querführung des Versuchsträgers automatisiert. Im Jahr 1996 veröffentlichte das Robotik Institut der Carnegie Mellon University einen Bericht über ihre „No Hands Across America"-Tour (automatische Fahrt von Washington, DC (USA) nach San Diego, CA (USA)) (Pomerleau u. Jochem, 1996). Dabei wurden 2796 Meilen der 2850 Meilen langen Strecke mit automatischer Querführung zurückgelegt (insg. 98,1 %).

Während der nächsten Jahre wurden eine Reihe von weiteren Projekten durchgeführt und so die einzelnen zum automatischen Fahren notwendigen Technologien immer weiter verbessert. Im Jahr 2003 rief die DARPA[2] den ersten DARPA Grand Challenge Wettbewerb aus (DARPA, 2003). Der Leistungsvergleich führte über eine Strecke von 150 Meilen durch die Mojave-Wüste und fand im Jahr 2004 statt. Dabei sollten die Versuchsträger die Strecke ohne Fahrer vollautomatisch zurücklegen. Von den 15 Teilnehmern schaffte es die Carnegie Mellon University als erfolgreichstes Team, eine Strecke von 7,3 Meilen zurückzulegen. Bei der im Jahr darauf stattfindenden 2. DARPA Grand Challenge wurde das Szenario nur leicht verändert (DARPA, 2005). Zusätzlich zum eigentlichen Wettbewerb gab es Vorausscheide, aus denen nur 23 der ursprünglich 40 Teams erfolgreich hervorgingen. Der eigentliche Wettbewerb erfolgte über eine Strecke von 132 Meilen und konnte von 6 Teams erfolgreich bewältigt werden (5 innerhalb des Zeitlimits). Die Serie der DARPA Grand Challenges wurde mit der DARPA Urban Challenge abgeschlossen (DARPA, 2006a). In diesem Wettbewerb wurde das Szenario von der Wüste in ein suburbanes Gebiet verlagert. Gegenüber den älteren Wettbewerben sollte hier eine Strecke von 60 Meilen zusammen mit anderen Verkehrsteilnehmern zurückgelegt werden. Dabei mussten die kalifornischen Verkehrsregeln eingehalten werden und in einer bestimmten Reihenfolge Punkte im Straßennetz angefahren werden. Von ursprünglich über 90 Teams wurden für den finalen Leistungsvergleich 11 Teams in mehreren Stufen ausgewählt. Von diesen Teams vervollständigten 6 Teams den Kurs, 4 innerhalb des Zeitlimits.

Während bei den DARPA Grand Challenges die Weiterentwicklung der Systeme durch das Ziel vollautomatischer militärischer Fahrzeuge motiviert war, wurden durch Fahrzeughersteller ab den späten 1990er und in den 2000er Jahren eine ganze Reihe von Fahrerassistenzsyste-

[1] Program for a European Traffic with Highest Efficiency and Unprecedented Safety
[2] Defense Advanced Research Projects Agency

men für den Endkunden verfügbar gemacht. So bietet Mercedes Benz seit 1999 eine adaptive Geschwindigkeitsregelanlage (ACC) mit aktivem Bremseingriff in seinen Fahrzeugen an (Winner u. a., 2009, Seite 479). Auf dieses System folgten eine Vielzahl weiterer, z. B.:

- Einparkassistenz mit aktiver Sensorik (Mercedes Benz, S-Klasse, 1995) (Reif, 2012, Seite 343)
- Fahrstreifenverlassungswarnung (Mercedes Benz, Actros, 2000) (Winner u. a., 2009, Seite 552)
- Nachtsichtassistenz (Cadillac, DeVille, 2000) (Winner u. a., 2009, Seite 465)
- Frontkollisionsschutz (Honda, Inspire, 2001) (Winner u. a., 2009, Seite 541)
- Fahrstreifenhalteassistenz (Honda, Nissa, 2002) (Winner u. a., 2009, Seite 554)
- Fahrstreifenwechselassistenz (Peugeot, Boxer, 2002) (Winner u. a., 2009, Seite 566)

Auch wenn diese Systeme zusammengenommen viele der für ein vollautomatisches Fahrzeug notwendigen Fähigkeiten bieten, sind vollautomatische Fahrzeuge noch nicht Teil des allgemeinen Straßenbildes. Die inzwischen im Markt befindlichen Assistenzfunktionen assistieren bei einzelnen Fahraufgaben und sind meist auf die Steigerung des Fahrkomforts und nicht als Sicherheitssystem ausgelegt. Aus diesem Grund weisen sie nicht die notwendige Zuverlässigkeit für eine alleinige Übernahme der Fahrfunktion ohne Zutun des Fahrers auf (Hoch- oder Vollautomation). Darüber hinaus sind die rechtlichen Rahmenbedingungen für den Regelbetrieb von (hoch-/voll-)automatisierten Fahrzeugen im deutschen Straßenverkehr für ein Endkundenprodukt nicht ausreichend geregelt.

Von 2010 bis 2012 beschäftigte sich die Projektgruppe „Rechtsfolgen zunehmender Fahrzeugautomatisierung“ (Bundesanstalt für Straßenwesen, 2012) mit der „Einschätzung der rechtlichen Situation bei zunehmender Fahrzeugautomatisierung im öffentlichen Straßenverkehr“ (Bundesanstalt für Straßenwesen, 2012, Seite 7). Hierzu wurden vier unterschiedliche fiktive Systeme unterschiedlichen Automatisierungsgrads auf die Übereinstimmung mit dem Straßenverkehrsrecht untersucht. Die Systeme wurden in fünf Klassen eingeteilt (siehe Glossar) deren Notation in dieser Arbeit genutzt wird. Die wesentlichen Unterschiede haben sich zwischen den Klassen Driver-Only bis teilautomatisiert und hoch- bis vollautomatisiert ergeben. Diese Unterscheidung basiert vor allem auf dem „erlaubten“ Verlust der Aufmerksamkeit des Fahrers bei hoch- und vollautomatisierten Systemen. Nach der StVO ist dieser Verlust nicht zulässig und steht somit dem Einsatz dieser hoch- und vollautomatisierten Systeme im Wege (Bundesanstalt für Straßenwesen, 2012, Seite 14f). Zur Fragestellung der Haftung nimmt die Arbeitsgruppe eine Einschätzung über die Aufteilung der Haftung dieser Systeme hinsichtlich Hersteller-, Fahrzeughalter- und Fahrerhaftung vor. Wie zu erwarten, steigt die Herstellerhaftung bei hoch- und vollautomatisierten Systemen, während sich für teilautomatisierte Systeme und darunter fast keine Änderung der Haftungsbedingungen gegenüber herkömmlichen Fahrzeugen ergeben.

2.1 Ausgewählte Projekte zum automatischen Fahren im städtischen Bereich

Im Folgenden werden Projekte zum hoch- und vollautomatisierten Fahren mit Forschungsaktivitäten im urbanen Bereich vorgestellt. Die Auswahl der Forschungsvorhaben resultiert auch aus der Verfügbarkeit von öffentlichen Informationen.

FU Berlin / AutoNOMOS Labs

Die Arbeitsgruppe von Raúl Rojas an der Freien Universität Berlin arbeitet seit einigen Jahren im Bereich des automatischen Fahrens. Mitte 2012 wurden drei Fahrzeuge von der Arbeitsgruppe betrieben. Der Versuchsträger „Sprit of Berlin“, ein Dodge Grand Caravan, wurde für den Einsatz in der DARPA Urban Challenge 2007 entwickelt (Rojo u. a., 2007) und mit einer Behindertenausstattung, DGPS sowie Laser- und Kamerasensoren ausgerüstet. Um die Richtlinien des Wettbewerbs zu erfüllen (DARPA, 2006b) und den vollautomatisierten Betrieb zu ermöglichen, wurde das Fahrzeug weiterhin mit einem Nothaltesystem ausgerüstet, das ein sofortiges Bremsmanöver von außen ermöglicht. Die Software dieses und der weiteren Fahrzeuge des Instituts basiert auf dem Open Robot Control Software Framework (Bruyninckx, 2001) und teilt sich in unterschiedliche Module, welche die Verarbeitung der Daten durchführen. Mit dem „Spirit of Berlin“ konnte das Team das National Qualification Event der DARPA Urban Challenge erreichen.

Nach der Teilnahme an der DARPA Urban Challenge wurde am Lehrstuhl ein weiteres Fahrzeug mit dem Namen „MadeInGermany“ entwickelt. Der Volkswagen Passat Variant ist ausgerüstet mit einem 360° Umfeldwahrnehmungssystem bestehend aus IBEO Lux-Lasersensoren, Radarsensoren der Hersteller TRW und SMS sowie einem 64-Ebenen Laserscanner des Herstellers Velodyne. Darüber hinaus kommen auch Stereo-Kamerasysteme sowie eine GPS-Inertialplattform mit Odometer zum Einsatz (Schwabe, 2012, Seite 5f). Der Versuchsträger wurde für die Fahrt im öffentlichen Straßenverkehr entwickelt. Aufgrund des im Fahrzeug anwesenden Sicherheitsfahrers ist das Projekt im teilautomatisierten Bereich anzusiedeln. Nach Wang u. a. hat das Fahrzeug seit Beginn des Projekts bereits über 4000 km auf öffentlichen Straßen zurückgelegt (Wang u. a., 2011). Dabei stehen zwei Einsatzbereiche im Vordergrund. Zunächst wurde die Fahrt auf Straßen höherer Kategorie betrachtet. So beschreibt Schwabe ausführlich die Verarbeitung der vorderen Radarsensordaten des Fahrzeugs und ihre Fusion mit den Sensordaten der Lasersensoren in einem Autobahnszenario (Schwabe, 2012). Die gewonnenen Erkenntnisse wurden anschließend auf ein Szenario im innerstädtischen Bereich übertragen. Hierzu wurden die Fähigkeiten des Versuchsträgers beispielsweise um eine kamerabasierte Fußgänger- und Lichtsignalanlagenwahrnehmung erweitert (Fassbender, 2012). Mitte September 2011 wurden die Ergebnisse in einer öffentlichen Testfahrt erfolgreich demonstriert (AutoNOMOUS Labs, 2011).

Neben dem automatischen Fahren beschäftigt sich die Gruppe am Versuchsträger „MadeInGermany“ auch mit der Handhabung von automatischen Fahrzeugen. So beschreibt Reuschenbach in seinen Arbeiten die Steuerung der Fahrzeuge „MadeInGermany“ und „Spirit of Berlin“ über ein Apple iPad bzw. ein Apple iPhone (Reuschenbach, 2011; Reuschenbach u. a., 2011). Diese erstreckt sich neben den sehr entwicklungsnahen Anwendungen wie Übertragung von Sensordaten oder direkte Steuerung von Gas und Lenkung auch auf eine Taxiapplikation, bei der ein Kunde das automatische Fahrzeug zu seinem Standort ruft. Eine weitere Entwicklung aus dem Bereich der Nutzerschnittstellen stellt das Projekt „BrainDriver“ dar (Waibel, 2011). Hierbei ist das Fahrzeug über Elektroden mit dem Fahrer verbunden. Dieser trifft an bestimmten Punkten im Straßennetz (z. B. Kreuzungen) Entscheidungen über den weiteren Verlauf der Fahrt.

Das neueste Fahrzeug der AutoNOMOS Labs ist das Fahrzeug „Einstein“ (Schwabe, 2012, Seite 6). Der Mitsubishi i-Mieve bietet als reines Elektrofahrzeug vor allem gute Einsatzmöglichkeiten im Indoorbereich. Der Versuchsträger verfügt über eine, gegenüber

den vorangegangenen Fahrzeugaufbauten, reduzierte Sensorausstattung. Dabei wurde die Menge an Umfeldwahrnehmungssensoren verringert und neue Herausforderungen in der Positionierung ohne GPS System angegangen. Als Anwendungsmöglichkeiten bieten sich hier Transportaufgaben von Personen oder Waren genauso an wie z. B. Reinigungsaufgaben in großen Hallen.

Stanford University

Nach der DARPA Urban Challenge wurde der Versuchsträger des Robotics Institut der Stanford University in diversen Bereichen weiterentwickelt (Levinson u. a., 2011a). Die Zielsetzung lag darin, die durch die Regeln der Urban Challenge vorgegebenen Beschränkungen weitestgehend zu überwinden und das Fahrzeug für den Betrieb im öffentlichen Straßenverkehr vorzubereiten. Neben der Umsetzung einer Verkehrszeichen- und Lichtsignalanlagenwahrnehmung (Levinson u. a., 2011b) wurde die Umfeldwahrnehmung verbessert. So beschreiben Teichman u. a. in ihrem Beitrag eine Klassifikation der Messpunkte eines Velodyne HDL64ES2 Laserscanners sowie das anschließende Tracking der Objekthypothesen (Teichman u. a., 2011). Ein weiterer Schwerpunkt liegt in der Optimierung der Fahrzeugpositionierung im städtischen Verkehr. Während in den Wettbewerben der DARPA Urban Challenge eine hochgenaue GPS-Position dauerhaft zu Verfügung stand, musste die Ortung nun um eine lokale Komponente erweitert werden. Als Grundlage dient dabei eine Gitterdatenstruktur mit der Infrarot-Reflektivität der Umgebung, die mithilfe eines SLAM-Algorithmus zu einer Position fusioniert wird (Levinson u. Thrun, 2010).

Google X

Die X Labs des Suchmaschinenanbieters Google beschäftigen sich ebenfalls sehr intensiv mit der Aufgabe des automatischen Fahrens. Nach der DARPA Urban Challenge gründete sich hier aus Mitgliedern der DARPA Challenge Teams der Carnegie Mellon University, University of Carlifornia, Berkely und der Stanford University ein Team um Sebastian Thrun. Im Oktober 2010 trat die Forschungsgruppe erstmals an die Öffentlichkeit (Thrun, 2010). In der veröffentlichten Pressemitteilung wurden teilautomatisierte Fahrten eines Toyota Prius im Bereich rund um San Francisco beschrieben. Insgesamt wurden durch verschiedene Versuchsträger mehr als 140.000 Meilen zurückgelegt. In einer Keynote zur IEEE International Conference on Intelligent Robots and Systems gab Thrun erstmals Informationen über die Verarbeitung von Daten und die Ausstattung der Versuchsträger (Guizzo, 2011). Sie sind ausgerüstet mit einem Velodyne HDL64ES2 Laserscanner, Kamerasystemen sowie Radarsensoren. Darüber hinaus ist jedes Fahrzeug mit einem GPS-System ausgerüstet. Der Laserscanner des Herstellers Velodyne stellt das Hauptsensorsystem dar. Die Kamerasysteme werden vor allem zur Wahrnehmung von bewegten Objekten (z. B. Fußgänger) genutzt. Zur Lokalisierung verwendet das Fahrzeug neben dem GPS-Sensor auch eine lokale Ortung über hochgenaue digitale Karten. Im Mai 2012 erhielt Google die Genehmigung des Verkehrsministeriums zum Betrieb von teilautomatischen Fahrzeugen im US-Staat Nevada (Nevada Department of Moto Vehicles, 2012). Dabei müssen während des Betriebs zu jedem Zeitpunkt zwei geschulte Personen mit gültiger Fahrerlaubnis im Fahrzeug anwesend sein.

Darüber hinaus ist über die Arbeit der Arbeitsgruppe von Thrun bei den Google X

Labs wenig Genaues bekannt. Die Presse schreibt der Gruppe großes Potential zu. So ist wohl auch das mediale Interesse an der von Sergey Brin veröffentlichten Ankündigung zur Teilnahme eines automatischen Rennwagens an der amerikanischen NASCAR-Serie zu verstehen (Brin, 2012). Diese entpuppte sich wenig später jedoch als April-Scherz, ohne dass viele der Presseartikel korrigiert wurden. Außerdem gibt es im Zusammenhang mit den Entwicklungen bei Google Bestrebungen Teile der Technologie zu patentieren. So wurde Google 2011 ein Patent auf den Betrieb von automatischen und manuell gefahrenen Fahrzeugen im gleichen Verkehrsraum zugesprochen (Schutz US, 8,078,349). Das Patent bezieht sich zwar nicht auf Kernkomponenten eines automatischen Fahrzeugs, sondern vor allem auf die Lokalisierung durch auf der Straße angebrachte Landmarken. Die Überschriften der Presse lauteten jedoch fälschlicherweise z. B. „Driverless car: Google awarded US patent for technology“ (BBC News, 2011).

Andere Gruppen

Neben den bereits erwähnten Gruppen gibt es einige, die sich nicht vorwiegend mit dem Thema des urbanen automatischen Fahrens befassen. Nach der DARPA Urban Challenge wurde in den Niederlanden die Cooperative Driving Challenge veranstaltet (TNO, 2009). Bei diesem Wettbewerb liegt die Zielsetzung in der Förderung des kooperativen Fahrens. Die Fahrzeuge sind dabei mit Car2Car und Car2Infrastructure Systemen ausgerüstet. Der Wettbewerb teilt sich in zwei Bereiche: ein Kreuzungsszenario sowie eine Fahrt auf einer Autobahn. Verglichen wird die Leistung zwischen den Teilnehmern anhand einer vordefinierten Bewertungsfunktion. Bei dem Wettbewerb am 14.-15. Mai 2011 bei Helmond, Niederlande, nahmen 11 Teams aus ganz Europa teil. Die Cooperative Driving Challenge gewann das Team AnnieWay vom Karlsruhe Institut of Technologie.

Von Juli bis Oktober 2010 führte das VisLab der Universität Parma die 2010 VisLab Challenge (VIAC) durch (Bertozzi u. a., 2011). Ziel war dabei die automatische Fahrt von Parma (IT) zur Expo 2010 in Shanghai (CN). Die Fahrt wurde als Kolonne zurückgelegt. Das Führungsfahrzeug zeichnete die aktuelle Straßenkarte auf und sendete diese an das automatische Folgefahrzeug. Die Versuchsträger waren dazu mit einer Vielzahl von Kamera- sowie Lasersensoren ausgestattet. Die Verarbeitung der Sensordaten basierte im Wesentlichen auf den Erfahrungen aus den DARPA Challenges 2004-2007 sowie dem Vorgängerfahrzeug „Braive“ (Bombini u. a., 2009) und wurde an die Anforderungen der VIAC angepasst. Während der Fahrt wurden die Versuchsträger mit unterschiedlichsten Straßen konfrontiert. Neben langen Strecken auf Autobahnen mussten auch Straßen ohne Belag und mit großen Schlaglöchern bewältigt werden. Die Verkehrssituation variierte von freier Fahrt bis hin zu starkem Verkehrsaufkommen in städtischen Umgebungen. Insgesamt legte die Kolonne 8244 km in automatischer Fahrt mit einer durchschnittlichen Geschwindigkeit von 38,4 $\frac{km}{h}$ zurück.

Nicht nur europäische und amerikanische Gruppen beschäftigen sich mit dem automatischen Fahren. So zeigte beispielsweise eine Forschergruppe um Prof. Dai Bin an der National Defense Technology University in Changsha nach Presseberichten eine vollautomatische unbemannte Fahrt von Changsha (CN) nach Wuhan (CN) (Nan, 2011). Dabei wurden 286 km auf einer Schnellstraße mit einer durchschnittlichen Geschwindigkeit von 87 $\frac{km}{h}$ zurückgelegt. Der Versuchsträger musste währenddessen 67 Überholmanöver selbstständig durchführen und mit Nebel und Gewittern zurechtkommen.

2.2 Sensorik zur Umfeldwahrnehmung

Im Automobilbereich haben sich für den Bereich Fahrerassistenzsysteme und den Bereich automatisches Fahren unterschiedliche Sensortechnologien zur Umfeldwahrnehmung etabliert. Je nach Anforderungen werden Radar-, Lidar-, Ultraschall-, Kamerasensoren bzw. Kombinationen dieser Technologien genutzt. Im Rahmen dieser Arbeit werden jedoch nur Radar- und Lidarsensoren eingesetzt. Die zugehörigen Messprinzipien werden im Folgenden kurz vorgestellt.

2.2.1 Lidarsensoren

Ein Lidarsensor (LIght Detection And Ranging) wird genutzt, um die Entfernung eines Objekts im Raum zu bestimmen. Eine ausführliche Einführung in die Thematik ist in Winner u. a. (2009, Seite 172ff) vorhanden. Das Verfahren basiert auf der Messung der Laufzeit t zwischen dem Senden eines Laserimpulses und dem Empfang der Reflexion von einer Reflektionsfläche durch eine Photodiode. Als Laser kommen ultraviolette, sichtbare oder infrarote Lichtquellen zum Einsatz. Die Entfernung d lässt sich mithilfe der Signallaufzeit t über die Lichtgeschwindigkeit c berechnen (Reif, 2012, Seite 333):

$$d = \frac{c \cdot t}{2} \tag{2.1}$$

Die Messung der relativen Geschwindigkeit der Reflexionsfläche zum Sender lässt sich über die Verschiebung der Frequenz des reflektierten Lichts messen (Dopplereffekt) (Reif, 2012, Seite 331):

$$f_{Doppler} = f_{Empfaenger} - f_{Sender} \tag{2.2}$$

$$f_{Doppler} = -\frac{2v_r \cdot f_{Sender}}{c} \tag{2.3}$$

Für einen typischen Lidarsensor im Infrarotbereich liegt die Sendefrequenz bei ca. 910 nm (329,44 THz). Die daraus resultierende Dopplerfrequenz $f_{Doppler}$ bei einer Annäherung an ein Objekt mit 20 $\frac{m}{s}$ beträgt damit ca. -43,9 MHz. Im Automobilbereich wird, bedingt durch diese Höhe der Frequenzen und die damit verbundenen Anforderungen an die Hochfrequenztechnik, aus Kostengründen auf die Messung der Dopplerfrequenz verzichtet.

Da ein Laserstrahl meist stark gebündelt ist, werden oft mehrstrahlige (Multi-beam) oder drehende Sensoren (Scanner) eingesetzt und so ein, gegenüber einem Single-beam-Sensor, größerer Erfassungsbereich ermöglicht. Multi-beam-Lidarsensoren ordnen die einzelnen Laser in unterschiedlichen Winkeln zueinander an. Bei Laserscannern lenkt meist ein drehender Spiegel den Laserstrahl in unterschiedliche Richtungen und tastet den Erfassungsbereich so zyklisch ab.

2.2.2 Radarsensoren

Die dominierende Sensortechnologie zur Umfelderfassung im Automobilbereich ist die Radar-Technologie (RAdio Detection And Ranging). Die meisten der derzeit in Serie verfügbaren ACC- und Notbremssysteme sind zumindest teilweise radarbasiert. Im Folgenden wird nur ein kurzer Überblick über die Technologie gegeben. Eine ausführliche Einführung findet sich in Winner u. a. (2009, Seite 123ff). Das Prinzip eines Radarsensors ist dem eines Lidarsensors ähnlich. Der Unterschied liegt in der Verwendung von elektromagnetischen Wellen einer

niedrigeren Frequenz. Zum Einsatz kommen im Automobilbereich vor allem Radarsensoren in den Frequenzbändern 24 GHz und 76-77 GHz.

Zur Messung sendet der Sensor eine elektromagnetische Welle aus und misst die Zeit t bis zum Empfang einer Reflexion von einem Hindernis. Mithilfe der Lichtgeschwindigkeit lässt sich so der Abstand berechnen (Reif, 2012, Seite 333):

$$d = \frac{c \cdot t}{2} \tag{2.4}$$

Durch die in Abschnitt 2.2.1 beschriebene Verschiebung der Frequenz durch einen Geschwindigkeitsunterschied zwischen Sender und Reflektor kann die Geschwindigkeit gemessen werden. Durch die bei Radarsensoren gegenüber Lidarsensoren verringerte Frequenz ist diese Messung wirtschaftlich machbar und bildet die Grundlage von vielen ACC-Systemen. Je nach Richtwirkung der Sende- und Empfangsantenne kann eine für entsprechende Anwendungen ausreichende Ortsauflösung erreicht werden. Diese wird jedoch meist durch einen geringeren Erfassungsbereich erreicht. Durch Rotation der Antennen oder durch Einsatz eines rotierenden Reflektors kann der Erfassungsbereich jedoch wieder erheblich erweitert werden.

Im Automobilbereich kommen vor allem zwei Radarmodulationsverfahren zum Einsatz. Das Pulsverfahren (vor allem 24 GHz) sendet kurze Impulse und misst anschließend die Reflektionen an den Objekten. Als zweites Verfahren kommt das Dauerstrichverfahren zum Einsatz. Hierbei wird ein kontinuierliches Signal mit wechselnder Frequenz ausgesendet (z. B. treppenförmig). Durch die Messung des Phasenverzugs zwischen Sender und Empfänger kann die Reflexionszeit t bestimmt werden.

2.3 Filter zur objekthypothesenbasierten Umfeldwahrnehmung

Je nach Anwendungsgebiet nutzen Fahrerassistenzsysteme unterschiedliche Zustandsgrößen eines Umfeld wahrnehmenden Systems. So benötigt beispielsweise eine ACC-Applikation den Abstand, die Geschwindigkeit und ggf. auch die Beschleunigung des vorausfahrenden Fahrzeugs (Winner u. a., 2009, Seite 478ff). Nicht alle diese Zustandsgrößen einer Objekthypothese können jedoch durch einen einzelnen Sensor gemessen werden. So kann ein Lidarsensor zwar den Abstand sehr genau bestimmen, Geschwindigkeit und Beschleunigung wird jedoch nicht gemessen (siehe Abschnitt 2.2.1). Ein Radarsensor kann zwar zusätzlich die Geschwindigkeit durch die Nutzung des Dopplereffekts bestimmen, die Beschleunigung ist jedoch ebenfalls nicht messbar.

Auch wenn diese Größen nicht messbar sind, so lassen sie sich doch anhand eines Modells aus den vorliegenden Messwerten bestimmen. Hierzu werden unterschiedliche Messungen in ihrem zeitlichen Zusammenhang betrachtet. Ist beispielsweise die Position einer Objekthypothese zu den Zeitpunkten k und $k+1$ bekannt, so kann daraus eine Geschwindigkeit abgeleitet werden, die die Objekthypothese zwischen den Zeitpunkten k und $k+1$ durchschnittlich hatte.

Da Messwerte meist fehlerbehaftet sind, ist die Berechnung der Geschwindigkeit v anhand des Weges Δd und der dafür benötigten Zeit Δt durch die Gleichung $v = \frac{\Delta d}{\Delta t}$ ebenfalls nicht exakt. Mit den Methoden der Fehlerfortpflanzung lässt sich bei Vorliegen der Unsicherheiten der Messwerte ein Geschwindigkeitswert mit einer Unsicherheit berechnen. Weil

der aktuelle Zustand einer Objekthypothese nur von dem vorherigen Zustand und dem aktuellen Messwert abhängt, liegt eine Markov-Kette 1. Ordnung vor (Bronstein u. a., 2001, Seite 786). Daraus folgend kann die Wahrscheinlichkeit für eine Zustandsschätzung als bedingte Wahrscheinlichkeitsfunktion $p(X[k+1]|X[k], Y[k+1])$ beschrieben werden. Zur algorithmischen Lösung der Berechnung werden im Folgenden einige verbreitete Verfahren vorgestellt. Da in dieser Arbeit ausschließlich diskrete Systeme betrachtet werden, werden im Folgenden jeweils die diskreten Varianten der Verfahren behandelt.

2.3.1 Kalman-Filter

Zur Lösung des Problems der Schätzung von nicht messbaren Zustandsgrößen wurden inzwischen eine Vielzahl von Methoden präsentiert. Eine Zusammenfassung vieler Verfahren findet sich in Blackman u. Popoli (1999). Im Folgenden wird speziell auf das Kalman-Filter (Kalman, 1960) bzw. das erweiterte Kalman-Filter (Cox, 1964) eingegangen, da es in dieser Arbeit verwendet wird. Das rekursiv angewandte Kalman-Filter bestimmt unter der Annahme eines linearen Systems, das mit weißem gaußschem Rauschen behaftetet ist, die optimale Zustandsschätzung im Sinne des mittleren quadratischen Fehlers (MSE).

Die folgende Gleichung beschreibt die Systemgleichung eines Modells, das einem diskreten Kalman-Filter zugrunde liegt:

$$\hat{x}[k+1] = F[k] \cdot \hat{x}[k] + q[k] + f[k+1|k] \tag{2.5}$$

$F[k]$ beschreibt das Systemmodell im Zustandsraum. $q[k]$ stellt den weißen gaußschen Rauschprozess mit der bekannten Kovarianz Q dar. Durch $f[k+1|k]$ werden deterministische Veränderungen gegenüber dem Prozessmodell abgebildet. Der Zustand eines Kalman-Filters wird zum Zeitpunkt k durch zwei Matrizen beschrieben: $\hat{x}[k]$ beschreibt den Zustandsvektor und $P[k]$ beschreibt die Kovarianzmatrix des Filters.

Der Filterprozess eines Kalman-Filters lässt sich in zwei Phasen aufteilen:

Prädiktion: In der Prädiktionsphase wird auf Basis der Systemgleichung der Zustandsvektor und die Kovarianzmatrix vom Zeitpunkt k auf den Zeitpunkt $k+1$ überführt. Hierzu wird das Systemmodell $F[k]$ auf den unprädizierten Zustandsvektor zum Zeitpunkt $k-1$ $\hat{x}[k-1|k-1]$ angewendet. Daran schließt sich die Prädiktion der Kovarianzmatrix $P[k-1|k-1]$ unter Verwendung des Systemmodells und der Kovarianzmatrix des Prozessrauschens $Q[k]$ an:

$$\hat{x}[k|k-1] = F[k] \cdot \hat{x}[k-1|k-1] \tag{2.6}$$

$$P[k|k-1] = F[k] \cdot P[k-1|k-1] \cdot F[k]^T + Q[k] \tag{2.7}$$

Korrektur: In der Korrekturphase wird der aktuelle Zustand des Kalman-Filters durch einen Messwert korrigiert. Dazu wird der Zustandsvektor $x[k|k-1]$ über die Messmatrix H auf den Messwert $z[k]$ abgebildet und daraus die Innovation $\tilde{y}[k]$ bestimmt. Anschließend wird die Kovarianzmatrix der Innovation $S[k]$ aus dem Messmodell, der Kovarianz des Filters und der Kovarianzmatrix des Messwerts berechnet. Die Bestimmung der Kalman-Verstärkung schließt sich an. Sie gibt einen Wert für die Übereinstimmung des Messdatums mit dem prädizierten Zustandsvektor $\hat{x}[k|k-1]$ an. Darauf folgt der Korrekturschritt des Zustandsvektors $\hat{x}[k|k]$ und die Aktualisierung der Kovarianzmatrix $P[k|k]$:

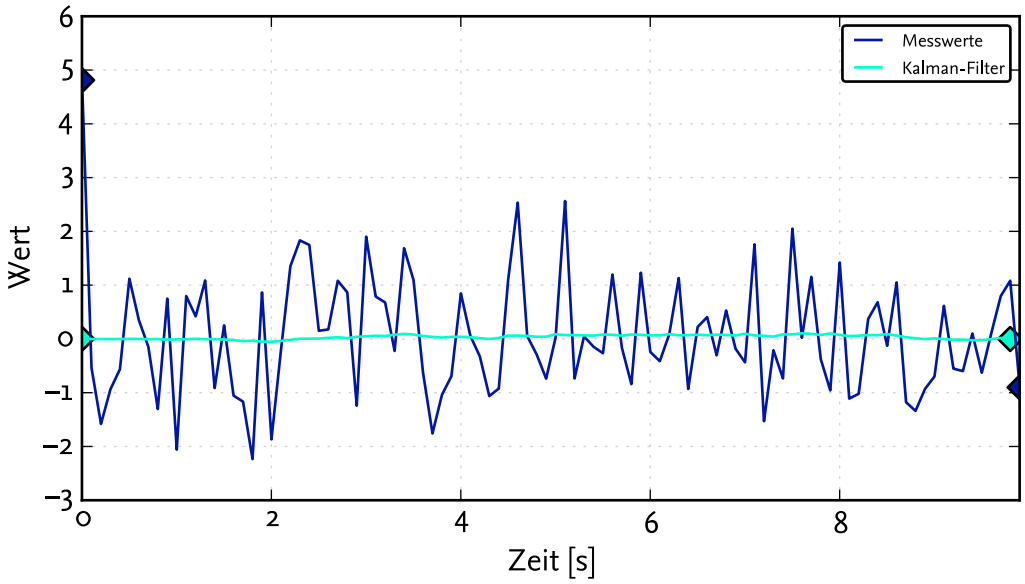


Abbildung 2.1: Schätzung einer Zufallsvariablen mit einem Kalman-Filter

$$\tilde{y}[k] = z[k] - H \cdot \hat{x}[k|k-1] \tag{2.8}$$
$$S[k] = H[k] \cdot P[k|k-1] \cdot H[k]^T + R[k] \tag{2.9}$$
$$K[k] = P[k|k-1] \cdot H^T S^{-1}[k] \tag{2.10}$$
$$\hat{x}[k|k] = \hat{x}[k|k-1] + K[k] \cdot \tilde{y}[k] \tag{2.11}$$
$$P[k|k] = (I - K[k] \cdot H[k]) \cdot P[k|k-1] \tag{2.12}$$

Die Herleitung der Formeln kann in Thrun (2010, Seite 40ff) nachvollzogen werden.

Im Folgenden wird das Kalman-Filter an einem Beispiel angewendet. Hierzu wird die Variable $\beta = 0$ geschätzt. Die Messwerte sind mit einem gaußschem weißen Rauschen behaftet. Die Standardabweichung der Messwerte beträgt $R = [1]$. Da in diesem Fall eine Konstante geschätzt wird, lautet das Prozessmodell $F[k] = [1]$. Die Anfangszustände des Filters werden mit den korrekten Werten initialisiert.

Zur Erzeugung des gaußschen weißen Rauschens mit $\sigma = 0$ wurde ein Pseudozufallsgenerator nach dem Standard FIPS 1862 eingesetzt (NORM FIPS 186-2, 2000). Dieser liefert zwar grundsätzlich gute Zufallszahlen, jedoch folgen diese als Pseudozufallszahlen nicht exakt dem gaußschem Rauschmodell. Der exakte Wert des Prozessrauschens ist nicht bekannt und wird deshalb geschätzt (Zhang, 1997, Seite 16). Das Prozessrauschen wird mit $Q = [0.0224^2] = [0.0005]$ angenommen.

In Abbildung 2.1 sind die Messwerte und das Ergebnis der Filterung zu erkennen. Dabei ist zu sehen, wie die Messwerte um den wahren Wert schwanken. Das Ergebnis des Filters folgt dem wahren Wert und weist nur geringe Abweichungen auf.

Wie anfangs erwähnt, ist für ein Kalman-Filter ein lineares System notwendig. Viele reale Systeme weisen jedoch nichtlineare Bestandteile auf. Aus diesem Grund schlug Cox eine

Erweiterung des Kalman Filters vor, die als erweitertes Kalman-Filter bekannt wurde (Cox, 1964). Zur Anwendung des erweiterten Kalman-Filters wird das Prozess- $F[k]$ und Messmodell $H[k]$ mithilfe einer Taylor-Reihe um den aktuellen Zustandsvektor $\hat{x}[k]$ linearisiert. Damit ergeben sich die Gleichungen für das linearisierte Prozess- $\tilde{f}(x[k])$ und Messmodell $\tilde{h}(x[k])$ unter Zuhilfenahme der zugehörigen Jacobimatrizen:

$$\tilde{f}(x[k]) \approx f(\hat{x}[k]) + F[k] \cdot (x[k] - \hat{x}[k]) \tag{2.13}$$

$$\tilde{h}(x[k+1]) \approx h(\hat{x}[k+1]) + H[k+1] \cdot (x[k] - \hat{x}[k]) \tag{2.14}$$

$$F[k] = \left.\frac{\partial f(x[k])}{\partial x[k]}\right|_{x[k]=\hat{x}[k]} = \left.\begin{bmatrix} \frac{f_1}{\partial x_1} & \cdots & \frac{f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{f_n}{\partial x_1} & \cdots & \frac{f_n}{\partial x_n} \end{bmatrix}\right|_{x[k]=\hat{x}[k]} \tag{2.15}$$

$$H[k] = \left.\frac{\partial h(x[k])}{\partial x[k]}\right|_{x[k]=\hat{x}[k]} = \left.\begin{bmatrix} \frac{h_1}{\partial x_1} & \cdots & \frac{h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{h_n}{\partial x_1} & \cdots & \frac{h_n}{\partial x_n} \end{bmatrix}\right|_{x[k]=\hat{x}[k]} \tag{2.16}$$

Da die Jacobimatrizen $F[k]$ und $H[k]$ um den jeweils aktuellen Zustandsvektor linearisiert werden, müssen sie in jedem Iterationsschritt neu berechnet werden. Hierdurch ergibt sich ein erhöhter Rechenaufwand. Die Formeln des Kalman-Filters ändern sich durch die Linearisierung nur leicht. Die geänderten Formeln lauten:

$$\hat{x}^{\cdot}[k|k-1] = \tilde{f}(x[k-1|k-1]) \tag{2.17}$$

$$P^{\cdot}[k|k-1] = F[k] \cdot P[k-1|k-1] \cdot F^T[k] + Q[k] \tag{2.18}$$

$$\tilde{y}^{\cdot}[k] = z[k] - \tilde{h}(x[k|k-1]) \tag{2.19}$$

$$S^{\cdot}[k] = H[k] \cdot P^{\cdot}[k|k-1] \cdot H^T[k] + R[k] \tag{2.20}$$

$$K^{\cdot}[k] = P^{\cdot}[k|k-1] \cdot H^{\cdot T}[k] S^{\cdot -1}[k] \tag{2.21}$$

$$\hat{x}^{\cdot}[k|k] = \hat{x}^{\cdot}[k|k-1] + K^{\cdot}[k] \cdot \tilde{y}^{\cdot}[k] \tag{2.22}$$

$$P^{\cdot}[k|k] = (I - K^{\cdot}[k] \cdot H[k]) \cdot P^{\cdot}[k|k-1] \tag{2.23}$$

Durch die Linearisierung des Prozess- und Messmodells treten offensichtlich Linierisierungsfehler auf. Diese Fehler sind abhängig von der Größe der Nichtlinearität und können in den Kovarianzmatrizen $Q[k]$ und $R[k]$ als Modellfehler modelliert werden.

2.3.2 Interacting-Multiple-Model-Filter

Dem Kalman-Filter liegt immer exakt ein Prozessmodell zugrunde. Diese Annahme ist in der Realität nicht immer gegeben. Bewegt sich beispielsweise ein Objekt mit einer konstanten Geschwindigkeit entlang einer Geraden, so lässt sich das Prozessmodell leicht herleiten. Kann sich das Objekt jedoch sowohl entlang einer Geraden als auch entlang eines Kreisbogens bewegen, so muss, um die Modellfehler zu minimieren, ein Prozessmodell gefunden werden, das beide Bewegungen abbildet. Dies ist nicht immer möglich oder ist aufgrund der daraus folgenden Komplexität des Modells nicht immer ratsam.

Eine Möglichkeit ist, das Objekt mit mehreren Filtern gleichzeitig zu verfolgen. Für diesen Zweck stellte Blom das Interacting-Multiple-Model-Filter (IMM) vor (Blom, 1984). In ihm

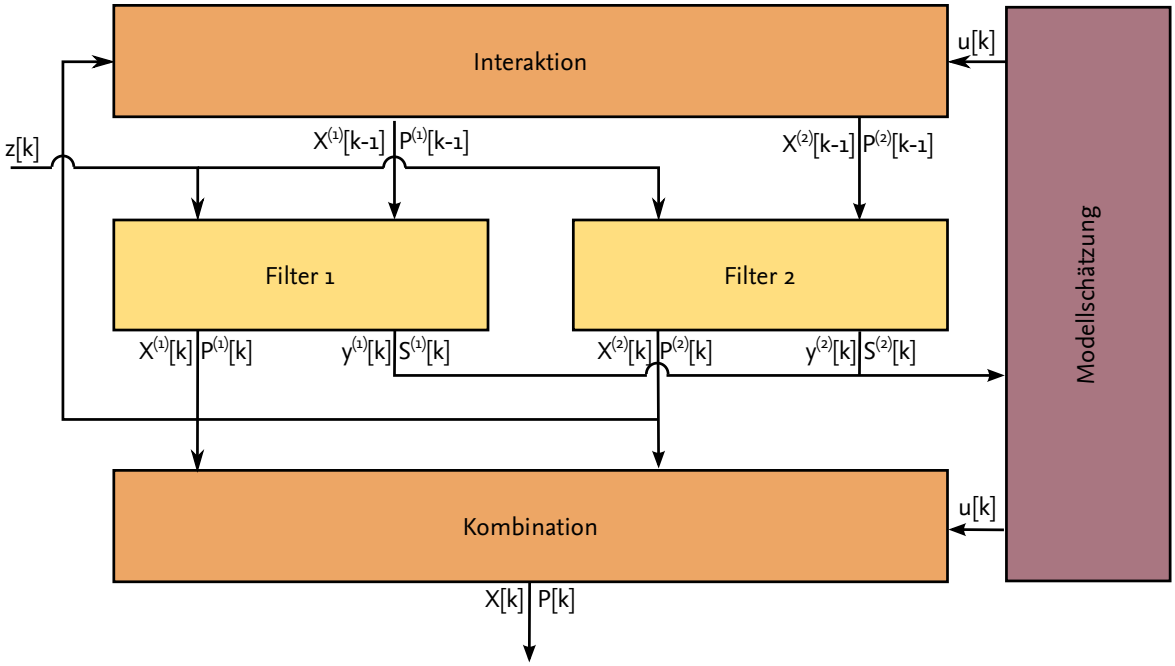


Abbildung 2.2: Ablauf des Filterprozesses eines Interacting-Multiple-Model-Filters mit zwei Filtern

werden mehrere getrennte Filter (nicht notwendigerweise Kalman-Filter) zu einem gemeinsamen System kombiniert und nach jedem Filterschritt ein kombinierter Zustandsvektor berechnet.

Der Filterprozess eines IMM-Filters ergänzt die genutzten Filter um drei Verarbeitungsschritte: Interaktion zwischen den Filtern, Kombination der Ergebnisse des Filters und Aktualisierung der Modellschätzung. In Abbildung 2.2 ist der Ablauf beispielhaft an einem IMM-Filter mit zwei inneren Filtern dargestellt. Zunächst wird anhand vorheriger Filterergebnisse und der aktuellen Wahrscheinlichkeitsverteilung für die einzelnen Modelle $u^{(i)}[k]$ für jeden Filter i ein Zustandsvektor $\bar{x}^{(i)}[k-1]$ und eine Kovarianzmatrix $\bar{P}^{(i)}[k-1]$ erstellt. Anschließend folgt ein Schritt des jeweiligen Filteralgorithmus (z. B. Kalman-Filter) unter Zuhilfenahme des aktuellen Messwerts $z[k]$. Mit der Abweichung $y^{(i)}[k]$ des Messwerts $z[k]$ vom prädizierten Zustand $\hat{x}^{(i)}[k|k-1]$ und der zugehörigen Kovarianzmatrix $S^{(i)}[k]$ kann nun die Modellschätzung aktualisiert werden. Anschließend werden die durch die Filter berechneten Zustandsvektoren $\hat{x}^{(i)}[k]$ und Kovarianzmatrizen $P^{(i)}[k]$ mithilfe der aktuellen Modellwahrscheinlichkeiten $u^{(i)}[k]$ zu einem Gesamtzustandsvektor $X[k]$ und einer Gesamtkovarianzmatrix $P[k]$ kombiniert. Die zugehörigen Formeln für ein IMM-Filter mit n Filtern sind wie folgt definiert (Blackman u. Popoli, 1999, Seite 225ff):

Interaktion: Im Interaktionsschritt werden die Zustandsvektoren $\bar{x}^{(i)}[k-1]$ und die Kovarianzmatrizen $\bar{P}^{(i)}[k-1]$ für die einzelnen Filter berechnet. Dazu sind neben den Zustandsvektoren und Kovarianzmatrizen des vorherigen Filterschritts die aktuellen Modellwahrscheinlichkeiten $u^{(i)}[k-1]$ und die Modellübergangswahrscheinlichkeitsmatrix $\check{P}$ nötig.

$$u^{ij}[k-1] = \frac{\check{P}^{ij}[k-1] \cdot u^{(i)}[k-1]}{C^j[k-1]} \tag{2.24}$$

$$C^j[k-1] = \sum_{i=1}^{r} \check{P}^{ij}[k-1] u^{(i)}[k-1] \tag{2.25}$$

$$\bar{x}^{(i)}[k-1] = \sum_{j=1}^{n} u^{ji}[k-1] \cdot \hat{x}^{(j)}[k-1] \tag{2.26}$$

$$\bar{P}^{(i)}[k-1] = \sum_{j=1}^{n} u^{ji}[k-1][P^{(j)}[k-1] + D \cdot \check{P}^{ji}[k-1]] \tag{2.27}$$

$$D \cdot P^{ji}[k-1] = D \cdot x^{ji}[k-1] \cdot D \cdot x^{ji}[k-1]^T \tag{2.28}$$

$$D \cdot x^{ji}[k-1] = \hat{x}^j[k-1] - \bar{x}^i[k-1] \tag{2.29}$$

Kombination: Zur Kombination der einzelnen Zustandsvektoren und Kovarianzmatrizen wird der nach den Modellwahrscheinlichkeiten gewichtete Mittelwert genutzt.

$$\hat{x}[k] = \sum_i u^{(i)}[k] \cdot \hat{x}^{(i)}[k] \tag{2.30}$$

$$P[k] = \sum_i u^{(i)}[k][P^{(i)}[k] + (\hat{x}[k] - \hat{x}^{(i)}[k])(\hat{x}[k] - \hat{x}^{(i)}[k])^T] \tag{2.31}$$

Modelschätzung: Für die Aktualisierung der Modellschätzung wird die Abweichung $y^{(i)}[k]$ des Messwerts zum prädizierten Zustandsvektor eines jeden Filters sowie dessen Kovarianzmatrix $S^{(i)}[k]$ benötigt.

$$d_i^2[k] = y^{(i)}[k]^T S^{(i)-1}[k] y^{(i)}[k] \tag{2.32}$$

$$\Lambda^i[k] = \frac{exp[-d_i^2[k] \cdot 0.5]}{\sqrt{(2\pi)^M |S^{(i)}[k]|}} \tag{2.33}$$

$$u^{(i)}[k] = \frac{\Lambda^i[k] \cdot C^i[k-1]}{\sum_{j=1}^{r} \Lambda^j[k] \cdot C^j[k-1]} \tag{2.34}$$

Haben die einzelnen Filter unterschiedlich Zustandsvektoren, erweitern sich die angegebenen Formeln um die entsprechenden Abbildungsfunktionen zwischen den Vektoren. Die Herleitung des IMM-Filters kann in Blom (1984) oder Blackman u. Popoli (1999, Seite 221ff) nachvollzogen werden. Ein Beispiel zur Anwendung des IMM-Filters findet sich in Abschnitt 16.2.

2.3.3 Weitere Filter

Zur Zustandsschätzung existieren eine Vielzahl weiterer Filtertechniken. So linearisiert das Unscented Kalman-Filter (Thrun, 2010, Seite 65ff) das Prozessmodell auf eine andere Weise als das erweiterte Kalman-Filter. Das Information-Filter stellt wiederum den Filterinhalt nicht im Zustands-, sondern im Informationsraum dar (Thrun, 2010, Seite 71ff). Diese und andere Filtertypen finden in der vorliegenden Arbeit keine Verwendung und werden daher an dieser Stelle nicht weiter vertieft.

2.4 Filter zur gitterbasierten Umfeldwahrnehmung

Die gitterbasierte Umfeldwahrnehmung wurde erstmals von Elfes vorgestellt (Elfes, 1990). Dabei wird nicht ein einzelnes wahrgenommenes Objekt durch einen Zustandsvektor beschrieben, sondern der beobachtete Bereich als Fläche von vielen einzelnen voneinander unabhängigen Zustandsvektoren (Gitterzellen) beschrieben. Diese Gitterzellen sind in einer äquidistanten Weise (bezogen auf eine mathematische Funktion) in einer Gitterdatenstruktur angeordnet. Zur Filterung der Messwerte haben sich zwei Filtertypen durchgesetzt: das Bayes-Filter und das Dempster-Shafer-Filter. Da in dieser Arbeit beide Filterverfahren für unterschiedliche Bereiche genutzt werden, wird auf beide Verfahren im Folgenden näher eingegangen.

2.4.1 Binäres Bayes-Filter

Das binäre Bayes-Filter basiert auf dem Satz von Bayes (Bronstein u. a., 2001, Seite 771):

$$P(B_j|A) = \frac{P(A|B_j) \cdot P(B_j)}{P(A)} = \frac{P(A|B_j) \cdot P(B_j)}{\sum_i P(A|B_i) \cdot P(B_i)} \tag{2.35}$$

Für ein binäres Bayes-Filter werden zwei Ereignisse angenommen x und $\bar{x}$. Eine Zelle enthält jeweils die Wahrscheinlichkeit für den Eintritt des Ereignisses x. Die Zellen werden für den Filterprozess als statistisch unabhängig voneinander betrachtet. Dies führt somit zum Vorliegen eines Markov Prozesses 1. Ordnung.

Betrachtet man den Anwendungsfall der gitterbasierten Umfeldwahrnehmung im Automobil, könnte das Ereignis x den Fall darstellen, dass eine Zelle durch den Versuchsträger befahren werden kann. Das Ereignis Y repräsentiert im Folgenden einen Messwert. Der Satz von Bayes lautet damit unter den gegebenen Annahmen für die Wahrscheinlichkeit des Ereignisses $x = bef. = befahrbar$ unter der Bedingung, dass das Ereignis Y eingetreten ist:

$$P[k](\text{bef.}|Y[k]) = \frac{P[k-1](\text{bef.}) \cdot P(Y[k])}{P(Y[k]) \cdot P[k-1](\text{bef.}) + (1 - P(Y[k])) \cdot (1 - P[k-1](\text{bef.}))} \tag{2.36}$$

2.4.2 Dempster-Shafer-Filter

Das binäre Bayes-Filter stößt bei der Verarbeitung von realen Messdaten teilweise an seine Grenzen, da die Unsicherheit eines Messwerts nicht ohne Weiteres abbildbar ist. So kann ein Messwert im binären Bayes-Filter nur ein Ereignis bzw. das Gegenereignis stützen. Bei einer ausreichenden Unsicherheit des Messwerts ist diese Aussage aber nicht haltbar.

Um dieser Herausforderung zu begegnen, kann die Evidenztheorie eingesetzt werden (Wu u. a., 2002). Sie bildet nicht nur das Vertrauen in Elementarereignisse ab, sondern kann zusätzlich auch das Vertrauen in Teilmengen vom Elementarereignissen abbilden. Eine Herleitung der Evidenztheorie auf der Basis der Wahrscheinlichkeitstheorie kann z. B. in Blackman u. Popoli (1999, Seite 521ff) gefunden werden.

Zur Beschreibung eines Problems mithilfe der Evidenztheorie wird zunächst ein Wahrscheinlichkeitspotenzraum $2^\Omega \rightarrow [0,1]$ bestehend aus Elementarereignissen gebildet. Für diesen Potenzraum gilt:

$$M(\emptyset) = 0 \tag{2.37}$$

$$\sum_{A \in \Omega} m[A] = 1 \tag{2.38}$$

Zur Anwendung für die gitterbasierte Umfeldwahrnehmung kann ein Wahrscheinlichkeitspotenzraum beispielsweise aus den Elementarereignissen *Befahrbar*, *NichtBefahrbar* und *Unbekannt* aufgebaut werden. Jede Gitterzelle enthält nun zwei Evidenzen (die Dritte kann berechnet werden). Aus den Messdaten für eine Gitterzelle wird ebenfalls eine Evidenz für jedes der Elementarereignisse bestimmt. Diese können nun über die Kombinationsregel von Dempster-Shafer miteinander fusioniert werden (Blackman u. Popoli, 1999, Seite 509):

$$M_C = M(C|A,B) = \frac{1}{1-\kappa} \sum_{i,j|A_i \cap B_j = C} M_A(A_i) \cdot M_B(B_j) \tag{2.39}$$

$$\kappa = \sum_{i,j|A_i \cap B_j = \emptyset} M_A(A_i) \cdot M_B(B_j) \tag{2.40}$$

Die Auswertung der Evidenzen für einzelne Elementarereignisse lässt sich anhand der Werte der zugehörigen Massen bestimmen. Soll jedoch das Vertrauen in ein Elementarereignis bestimmt werden, so sind u. U. auch andere Evidenzen von Bedeutung. Besteht der Wahrscheinlichkeitspotenzraum beispielsweise aus den Ereignissen *Befahrbar*, *NichtBefahrbar* und *Unbekannt*, so ist zur Berechnung des Vertrauens in das Elementarereignis *Befahrbar* auch die Masse des Ereignisses *Unbekannt* notwendig. Mithilfe der Support- und Plausibilitäts-Funktion kann je die untere und obere Schranke dieses Vertrauens bestimmt werden (Blackman u. Popoli, 1999, Seite 510):

$$Spt(\bar{X}|A,B) = \frac{1}{1-\kappa} \sum_{i,j|A \cap B_j \subseteq X} m_A(A_i) m_B(B_j) \tag{2.41}$$

$$= \sum_{C \Big| \substack{C \in \mathbb{C} \\ C \subseteq X}} m(C|A,B) \tag{2.42}$$

$$Pls(\bar{X}|A,B) = \frac{1}{1-\kappa} \sum_{i,j|(A \cap B_j) \cap X \neq \emptyset} m_A(A_i) m_B(B_j) \tag{2.43}$$

$$= \sum_{C \Big| \substack{C \in \mathbb{C} \\ C \cap X \neq \emptyset}} m(C|A,B) \tag{2.44}$$

$$= 1 - Spt(\bar{X}|A,B) \tag{2.45}$$

Die Support-Funktion $Spt(\bar{X}|A,B)$ drückt dabei das Gesamtmaß an Vertrauen in ein Elementarereignis aus. Die maximale Evidenz in das Ereignis lässt sich wiederum mit der Plausibilitäts-Funktion $Pls(\bar{X}|A,B)$ beschreiben.

2.5 Architekturen zur Umfeldwahrnehmung

Unter dem Begriff der Architektur zur Umfeldwahrnehmung oder Sensordatenfusionsarchitektur werden in der Literatur unterschiedliche Konzepte beschrieben. Die erste Gruppe von Beiträgen beschreibt unter diesen Überschriften, welche Sensoren innerhalb eines Projekts genutzt werden, wie die Messdaten anschließend verarbeitet und letztlich einer Applikation zur Verfügung gestellt werden (Al-Dhaher u. Mackesy, 2004; Durrant-Whyte u. a., 1990;

Effertz, 2008). Die dort beschriebenen Architekturen zeigen Möglichkeiten der Anwendung auf. Sie lassen sich jedoch nur bedingt verallgemeinern.

Eine weitere Gruppe beschreibt allgemein den Aufbau von Sensordatenfusionssystemen. Sie lässt sich weiter unterteilen. In dieser Gruppe existiert eine Menge von älteren Veröffentlichungen, die als Grundstock für spätere Beschreibungen dienen. So veröffentlichte Boyd eine grundsätzliche Abfolge von Aktionen zur Verarbeitung von Informationen und die daraus erfolgende Ableitung einer Entscheidung (Boyd, 1976). Im Jahr 1997 veröffentlichte Dasarathy das Waterfall-Modell (Markin u. a., 1997; Dasarathy, 1997). Es beschreibt eine Fusionsarchitektur als linearen Ablauf von wenigen abstrakten Verarbeitungsschritten. Grundsätzlich entspricht diese Verarbeitungskette der aller modernen Sensordaten(fusions)systeme. Das Omnibus-Modell der Autoren von Bedworth u. O'Brien entwickelt die vorangegangenen Modelle weiter (Bedworth u. O'Brien, 1999). Der Ablauf der Verarbeitung lehnt sich an das Modell von Boyd an, interpretiert es allerdings in Bezug auf die Anwendung im Bereich der Datenfusion. Alle diese Beiträge geben die grobe Struktur eines Systems vor. Sie sind jedoch durch einen hohen Abstraktionsgrad in ihrer Anwendung limitiert bzw. geben dem Entwickler keine konkrete Anleitung zur Strukturierung seiner Module an die Hand. Eine Übersicht über diese und weitere Architekturen findet sich z. B. in Gad u. Farooq (2002) oder Bedworth u. O'Brien (1999).

Einige der neueren Veröffentlichungen, insbesondere mit Bezug zum automobilen Bereich, werden im Folgenden vorgestellt. Am Ende des Abschnitts folgt eine Einordnung dieser Beiträge in den Kontext dieser Arbeit.

LAAS-Architektur

Anders als die bisher vorgestellten Architekturen geht die LAAS-Architektur konkreter auf die zu implementierenden Komponenten und ihr Zusammenspiel ein. Die LAAS-Architektur (Laboratoire d'Analyse et d'Architecture des Systèmes) wurde als allgemeine Architektur zur Steuerung von Robotern entwickelt (Alami u. a., 1998). Sie umfasst, wie die anderen vorgestellten Architekturen, nicht nur den Bereich der Sensordatenverarbeitung, sondern auch den Bereich der Entscheidungsfindung und der Entscheidungsausführung. Die Architektur beschreibt fünf Ebenen: physisches System, logische Roboterebene, funktionale Ebene, Ausführungs-/Regelebene und Entscheidungsebene. Innerhalb jeder Ebene werden die Aufgaben beschrieben und die Organisation der Module festgelegt. Die LAAS-Architektur fand jedoch keine größere Verbreitung außerhalb der Forschungseinrichtung, da die nun folgende JDL-Architektur bereits die Aufgabe einer Grundlagenarchitektur übernommen hatte.

JDL-Architektur

Die JDL-Architektur wurde im Auftrag des US Department of Defence von den Joint Directors of Laboratories'(JDL) Data Fusion Sub-Panel entwickelt (White (1988) zitiert nach Steinberg u. a. (1999)). In Abbildung 2.3 ist das Fusionsmodell dargestellt. Es teilt sich in fünf Ebenen auf:

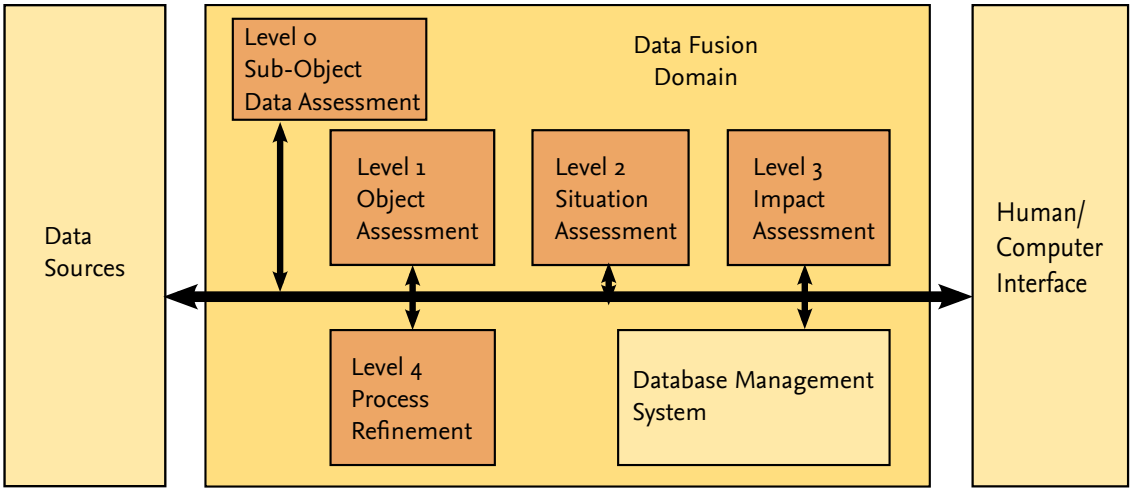


Abbildung 2.3: JDL Datenfusionsmodel nach Steinberg u. a. (1999, Seite 6)

Level 0 - Sub-Object Data Assessment: Das Level 0 führt auf den Datenquellen eine Vorverarbeitung auf Signal-/Messwertebene durch.

Level 1 - Object Assessment: In dieser Ebene werden die vorverarbeiteten Messdaten durch Assoziation mit bereits bekannten Objekten und anschließendem Tracking in ein allgemeines Datenformat (Tracks) überführt.

Level 2 - Situation Assessment: Die zuvor erstellten Tracks werden nun in Beziehung zueinander gesetzt und eine Situationsanalyse durchgeführt.

Level 3 - Impact Assessment: Anschließend werden die Situationen in Hinblick auf die aktuellen Missionsziele bewertet.

Level 4 - Process Refinement: Level 4 plant das weitere Vorgehen und setzt dieses um. Hierzu können die Algorithmen anderen Aufgaben erteilen und können so auch für eine Verteilung von Ressourcen innerhalb des Systems sorgen.

Wie in Abbildung 2.3 dargestellt, sind die Ebenen nicht zwangsläufig als Hierarchie zu verstehen. Verbunden sind sie über einen gemeinsamen Bus, über den auch weitere Infrastruktur (z. B. Datenbanken oder HMI) angesteuert werden kann. Durch die Nutzung einer Businfrastruktur ist es möglich einzelne Ebenen zu überspringen, sollten diese für eine konkrete Aufgabe nicht benötig werden (Steinberg u. a., 1999, Seite 5).

Das JDL-Modell hat die Grundlage für viele Architekturen zur Sensordatenfusion gelegt und bildet für aktuelle Architekturen immer noch die Basis. Es wurde in den letzten Jahren immer wieder kritischen Untersuchungen unterzogen. So begannen die ursprünglichen Autoren in Steinberg u. a. (1999), einige Missverständnisse in der Anwendung des Modells richtigzustellen. In Llinas u. a. (2004) wurden zusätzliche Erweiterungen vorgestellt. So wurde vorgeschlagen die 4. Ebene aus dem Fusionsprozess zu entfernen und weiter auf die Zusammenarbeit der verbleibenden Ebenen, u. a. in Bezug auf verteilte Systeme, eingegangen. Als 5. optionale Ebene wurde der Bereich des Human Machine Interfaces eingeführt, der im ursprünglichen Modell noch außerhalb der Data Fusion Domain angesiedelt war.

ProFusion2-Architektur

Eine Anpassung des JDL-Modells auf den Automobilbereich ist in Polychronopoulos u. Amditis (2006) dargestellt. Für das EU-Projekt PReVENT wurde im Teilprojekt ProFusion die ProFusion2-Architektur (PF2) entwickelt. Innerhalb der PF2-Architektur wurden drei Ebenen definiert: die Wahrnehmungsebene kombiniert die Ebenen 0 und 1 der JDL-Architektur, die Entscheidungs-Anwendungs-Ebene enthält die Ebenen 2 und 3 und die Aktions-/HMI-Ebene bildet die Ebene 5 ab. Innerhalb der Wahrnehmungsebene wurde die JDL-Ebene 1 in drei einzelne Komponenten aufgeteilt. Sie repräsentieren unterschiedliche Abstraktionsgrade der Messdaten (Feature, Track, Objekt). Die Kategorisierung lehnt sich an das in Scheunert u. a. (2006) beschriebene Extended 4D Environment Model an. Die PF2-Architektur wurde im Projekt PReVENT bei fünf unterschiedlichen Sensordatenfusionssystemen angewendet (Tatschke u. a., 2006): Der Early Fusion Approach fusioniert direkt die Rohdaten der eingesetzten Sensoren. Im Multi Level Fusion Approach wurden die wahrgenommenen Objekte jedes einzelnen Sensor zunächst in einer eigenen Sensordatenfusion getrackt und anschließend in einer gemeinsamen Sensordatenfusion fusioniert. Der Grid-based Fusion Approach fusioniert die Messdaten zunächst in einer Gitterdatenstruktur und wendet anschließend Algorithmen zur Objekthypothesenbildung an. Mit dem Track-Level Fusion Approach wurde die Anwendung der Architektur auf die Fusion bereits gebildeter Objekthypothesen untersucht. Der letzte Ansatz ist der Fusion Feedback Approach. Hierbei wurde die Möglichkeit untersucht, Tracks zum Zwecke der Objekthypothesenbildung rückzukoppeln.

Basissystemarchitektur zur Sensordatenfusion nach Darms (2007)

Im Projekt PRORETA 1 (PRORETA, 2002) wurde eine Basissystemarchitektur zur Sensordatenfusion von Umfeld Sensoren entwickelt (Darms u. Winner, 2005; Darms, 2007). Diese Architektur richtet sich ausschließlich an objekthypothesenbasierte Fusionssysteme und teilt sich in drei Ebenen auf: Die Sensorebene übernimmt die Verarbeitung der Messdaten, die Ableitung der daraus entstehenden Merkmale, ihre Assoziation sowie eine auf die Messdaten zugreifende Klassifikation. In der Fusionsebene wird eine klassische objekthypothesenbasierte Sensordatenfusion umgesetzt. Zusätzlich wird auch der Bereich der Objekthypothesenklassifikation sowie die Möglichkeit der Priorisierung von Objekthypothesen durch eine Anwendung hier angesiedelt. Abgeschlossen wird die Architektur mit der Anwendungsebene, die eine Fahrerassistenzfunktion abbildet.

Die Architektur von Darms beschreibt die Strukturen wie alle vorangegangenen Architekturen vor allem aus der Laufzeitsicht. Die Schnittstellen zwischen den Ebenen werden zwischen der Sensorebene und der Fusionsebene durch ein sensorspezifisches Datenformat abgebildet. Zwischen der Fusionsebene und der Anwendungsebene wird ein anwendungsspezifischer Szenenbaum genutzt, in dem Objekte über eine Ableitungs- und Assoziationsstruktur abgebildet werden. Da in PRORETA der Längsverkehr im Focus stand, wurde hier die Klasse Fahrzeugheck als Ableitung eines allgemeinen Polygonzugs genutzt. Auf die Schnittstellen zwischen den Komponenten der einzelnen Ebenen wird in der Arbeit nicht im Allgemeinen, sondern nur in Bezug auf die spätere Umsetzung näher eingegangen.

Generische Sensordatenfusion nach Munz (2011)

Das Institut für Mess-, Regel- und Mikrotechnik der Universität Ulm kann auf eine lange Erfahrung mit Umfeld wahrnehmenden Systemen zurückblicken. So beschrieben Dietmayer u. a. in einem wissenschaftlichen Beitrag grundlegende Architekturprinzipien zur Sensordatenfusion (Dietmayer u. a., 2005). Dabei wurde vor allem auf die objekthypothesenbasierte Fusion eingegangen und grundlegende Definitionen zu Sensordatenfusionssystemen durchgeführt. Angewendet wurden diese Überlegungen am Beispiel einer Fusionsarchitektur der Volkswagen Forschung (Dietmayer u. a., 2005, Seite 76ff).

Die Idee einer allgemeinen Sensordatenfusion wurde am Lehrstuhl weiter verfolgt. Im Projekt generische Sensordatenfusion wurde eine applikationsunabhängige Datenfusion entwickelt und von Munz ausführlich vorgestellt (Munz, 2011). Munz beschreibt insgesamt sechs notwendige Schritte für eine Sensordatenfusion zzgl. der verarbeitenden Applikation: Zunächst wird eine Messdatenerfassung durchgeführt, anschließend findet die Signalvorverarbeitung statt. Diese wird von einer Merkmalsextraktion gefolgt und durch eine Datenassoziation und die Aktualisierung der Objektverfolgung verarbeitet. Darüber hinaus werden die Daten durch eine Objektverwaltung verarbeitet und schließlich einer Applikation übergeben. Die Herausforderung der Applikationsunabhängigkeit wird in der Arbeit durch die Nutzung einer gleichzeitigen Zustands- (Kämpchen, 2007) und Existenzschätzung (Mählisch, 2009) realisiert. Durch die Existenzinformation einer Objekthypothese können unterschiedliche Applikationen ganz nach ihren Anforderungen auch auf unsichere Objekthypothesen reagieren oder diese zunächst ignorieren und warten, bis eine ausreichende Wahrscheinlichkeit für deren Existenz vorliegt. Zur Austauschbarkeit von Sensoren wurde zwischen Sensormodulen (Schritt 1-3) und den Fusionsmodulen (Schritt 4-6) eine sensorunabhängige Schnittstelle beschrieben. Diese besteht neben dem Zustandsvektor einer Objekthypothese aus deren statistischen Eigenschaften (z. B. Detektions- oder Rückschlusswahrscheinlichkeit).

General Data Fusion Architecture

Carvalho u. Heinzelman stellten 2003 eine allgemeine Datenfusionsarchitektur vor. Dabei gingen sie vor allem auf die statische Architektursicht ein. In ihrem Beitrag (Carvalho u. Heinzelman, 2003) definierten sie drei unterschiedliche Klassen von Fusionssystemen: lowlevel Fusionssysteme verarbeiten nur wenig vorverarbeitete Messdaten. Highlevel Fusionssysteme verarbeiten Daten, welche vorher bereits durch lowlevel Fusionssysteme und eine Datenanalyse verarbeitet wurden. Die dritte Klasse stellt eine Kombination aus den beiden anderen Fusionssystemen dar. Als Anwendungsbereich für die Architektur sehen sie eine breite Palette an Möglichkeiten. Durch die sehr abstrakte Beschreibung der Architektur beschreiben sie Einsatzmöglichkeiten im Spektrum von militärischen Anwendungen bis zur Überwachung von medizinischen Daten.

Kritische Betrachtung und Bezug zur vorliegenden Arbeit

Im Bereich der Architektur zur Umfeldwahrnehmung wurden bereits einige Arbeiten vorgelegt. Die älteren Arbeiten wie (Boyd, 1976; Dasarathy, 1997; Bedworth u. O'Brien, 1999) bilden eine gute Grundlage für weiterführende Arbeiten. Sie sind durch ihren hohen

Abstraktionsgrad allerdings nur bedingt auf konkrete Probleme anwendbar. Das Gleiche lässt sich auch über die von Carvalho u. Heinzelman vorgestellte Architektur sagen. Die LAAS-Architektur fand keine größere Verbreitung. Dies wurde sicher auch dadurch bedingt, dass die JDL-Architektur den Platz als allgemeine Referenzarchitektur bereits eingenommen hatte. Für den Automobilbereich wurde mit der Arbeit von Darms eine viel beachtete Arbeit veröffentlicht. Sie betrachtet jedoch nur die Laufzeitsicht einer Architektur und bietet auch keine Möglichkeit effektiv andere Fusionsverfahren (z. B. gitterbasierte Fusion) mit zu unterstützen. Die PF2-Architektur unterstützt zwar sowohl gitter- als auch objekthypothesenbasierte Fusionen, wird jedoch auch auf einem sehr hohen Abstraktionsgrad beschrieben. Die Anwendbarkeit liegt hier, ähnlich wie bei der JDL-Architektur, vor allem in der vorgegebenen Struktur. Die Arbeiten von Munz gehen im Vergleich zu der vorliegenden Arbeit einen anderen Weg. Bei Munz wird der Gedanke verfolgt, mit ein und derselben Sensordatenfusion alle Applikationen bedienen zu können. In der hier vorliegenden Arbeit wird jedoch ein System beschrieben, mit dem auf einfache Weise eine spezifische Fusion für eine oder mehrere spezielle Applikationen erstellt werden kann.

3 Forschungsvorhaben Stadtpilot

Das Projekt Stadtpilot ist ein Forschungsvorhaben der TU Braunschweig (Wille u. a., 2009). Es wird derzeitig vom Institut für Regelungstechnik sowie dem Institut für Flugführung im Themenbereich „Intelligentes Fahrzeug“ des Niedersächsischen Forschungszentrums Fahrzeugtechnik durchgeführt. Im Frühjahr 2008 wurden die ersten Gespräche zur Definition des Projektrahmens und -ziels sowie zur Zusammensetzung des Konsortiums geführt und das Vorhaben im Sommer 2008 offiziell begonnen. Im Herbst 2010 wurden die ersten Ergebnisse der Öffentlichkeit unter großer Anteilnahme der Medien vorgestellt (Technische Universität Braunschweig, 2010; Pudenz, 2010; Neue Zürcher Zeitung, 2010). Seitdem wird das Projekt weiterentwickelt und neue Funktionalitäten werden hinzugefügt. Der folgende Abschnitt beschreibt die Projektziele bis Herbst 2012. Diese wurden inzwischen weiterentwickelt und werden im Folgenden nicht beschrieben, da sie für diese Arbeit keine Relevanz mehr besitzen.

Hervorgegangen ist das Projekt aus dem Team CarOLO (Rauskolb u. a., 2008) der DARPA Urban Challenge. Hier nahm das Team als erfolgreichstes nicht amerikanisches Team an den Wettbewerben teil und fuhr zusammen mit 10 anderen Teams im Finale des Wettbewerbs. Neben einigen Sensoren des Versuchsträgers wurde vor allem die Erfahrung aus dem Vorgängerprojekt transferiert. So baut beispielsweise die Umfeldwahrnehmung auf den Erfahrungen von Effertz (2008) und Ohl (2007) auf. Auch konnte bei den Regelalgorithmen und der Ansteuerung des Fahrzeugs auf die Erfahrungen aus der DARPA Urban Challenge zurückgegriffen werden.

3.1 Projektziel

Das Ziel des Projekts Stadtpilot war die teilautomatische Umrundung des Braunschweiger Stadtkerns auf der umlaufenden Ringstraße (siehe Abbildung 3.1). Dabei sollte sich sich das Fahrzeug teilautomatisch im fließenden öffentlichen Verkehr bewegen und musste somit alle geltenden Verkehrsregeln beachten und auch mit unerwarteten Situationen umgehen können. Der Versuchsträger sollte während der Fahrt durch einen Computer gesteuert werden. Die Überwachung führte ein Sicherheitsfahrer durch. Er war jederzeit in der Lage die Fahrzeugführung zu übernehmen, sollte die Situation dies erfordern.

Das geplante Szenario sah den Beginn auf dem Parkstreifen auf der Hans-Sommer-Straße, Richtung Westen, vor (siehe Abbildung 3.1, Punkt 1). An dieser Stelle übernimmt das System das Ausparken und Einfahren in den fließenden Verkehr. Über mehrere große Kreuzungen mit Lichtsignalanlagen führt die Fahrt den Ring entlang. Während der Fahrt wird dem Fahrstreifen gefolgt und ggf. ein Fahrstreifenwechsel durchgeführt. Nach ca. 40 % des Szenarios muss an einer Lichtsignalanlage ein Linksabbiegemanöver durchgeführt werden (siehe Abbildung 3.1, Punkt 2), ohne dass dieses Manöver durch die Lichtsignalanlage explizit geregelt wird. Anschließend wird die Fahrt bis zur Hans-Sommer-Straße, Richtung Osten, fortgesetzt und auf einen Parkplatz (siehe Abbildung 3.1, Punkt 3) eingebogen. Abgeschlossen wird das Szenario durch die Identifikation eines freien Parkplatzes und dem sich anschließenden Einparkmanöver.

Im Oktober 2010 wurde nur ein Teilumfang des Szenarios auf einem $2,5km$ langen Teilstück

Abbildung 3.1: Karte des Stadtpilot-Szenarios
(Geodaten ©OpenStreetMap & Mitwirkende, CC BY SA)

gezeigt. Dieses umfasste die Übernahme der Kontrolle durch das System im fließenden Verkehr auf dem Rebenring, Richtung Osten, (siehe Abbildung 3.1, Punkt 4) und das anschließende Folgen des Fahrstreifens. Auf der Hans-Sommer-Straße (siehe Abbildung 3.1, Punkt 1) wurde das Fahrzeug manuell gewendet, um seine Fahrt fortzusetzen. An der Kreuzung Rebenring, Ecke Mühlenpfordstraße (siehe Abbildung 3.1, Punkt 5) führte der Versuchsträger ein automatisches Wendemanöver aus und beendete anschließend seine Fahrt wieder am Startpunkt. Die Erkennung von Lichtsignalanlagen wurde während der Fahrt durch den Beifahrer realisiert.

Das im Herbst 2012 geplante Szenario umfasste bereits einen erheblich erweiterten Funktionsbereich. Begonnen wird die Fahrt erneut im fließenden Verkehr (siehe Abbildung 3.1, Punkt 1). Anschließend wird dem Fahrstreifen bis zum Rudolfsplatz gefolgt und ggf. ein Fahrstreifenwechsel durchgeführt. Der Rudolfsplatz stellt den Wendepunkt der Route dar (siehe Abbildung 3.1, Punkt 6). Anschließend folgt der Versuchsträger erneut dem Fahrstreifen bis zur Hans-Sommer-Straße, Richtung Osten (siehe Abbildung 3.1, Punkt 3). Die

Wahrnehmung des Signalbilds der Lichtsignalanlagen wird in diesem Szenario bei drei Lichtsignalanlagen automatisch durchgeführt, bei anderen Lichtsignalanlagen wird das Signalbild durch den Beifahrer wahrgenommen und in das System eingegeben.

3.2 Fahrzeugumfeld

Das Fahrzeugumfeld im Projekt Stadtpilot zeichnet sich durch große mehrspurige Straßen aus. Die Fahrtrichtungen auf diesen Straßen sind auf großen Teilen des Szenarios baulich voneinander getrennt. In einem geringen Teil befindet sich zwischen den Fahrtrichtungen lediglich eine durchgezogene Linie als Trennung. Die gesamte Strecke ist mit hinreichend guten Fahrstreifenmarkierungen versehen und durchgängig an allen Kreuzungen mit Lichtsignalanlagen ausgestattet. Im Frühjahr 2012 waren drei dieser Lichtsignalanlagen im Rahmen des Forschungsvorhabens „Anwendungsplattform Intelligente Mobilität“ (Köster u. Frankiewicz, 2012) mit Sendern ausgestattet, die den aktuellen Zustand sowie die Kreuzungstopologie drahtlos übermitteln.

Die Verkehrsteilnehmer auf der Strecke bestehen vorwiegend aus anderen PKW mit einem geringen Anteil von Bussen, LKW sowie Motorrädern. Das Verkehrsaufkommen schwankt je nach Tageszeit zwischen Stopp & Go Verkehr und freier Fahrt mit einer Geschwindigkeit von bis zu $60\ \frac{km}{h}$. Fußgänger und Fahrradfahrer halten sich nur in Ausnahmefällen auf der Fahrbahn auf. In der Regel kreuzen sie die Fahrbahn nur an mit Lichtsignalanlagen ausgestatteten Stellen. Die Fahrstreifenbreite beträgt ca. $3,1m$ (Saust u. a., 2010) und bietet damit nur sehr geringe Sicherheitsabstände zu den Seiten.

Bedingt durch das städtische Umfeld existiert entlang der Strecke eine große Menge Randbebauung sowie auf großen Teilen abgestellte Fahrzeuge an den Fahrbahnrändern. Zusätzlich zu Gebäuden sind große Teile der Strecke durch Baumbewuchs abgeschattet, was zu Licht- und GPS-Schatten führt.

Das Ziel der Szenarios liegt auf einem Parkplatz. Dieses Gebiet unterscheidet sich erheblich von dem des Stadtrings. Es zeichnet sich durch i. d. R. sehr geringe Sicherheitsabstände sowie sehr geringe Geschwindigkeiten aus. In Bereich des Parkplatzes ist mit Fußgängern genauso zu rechnen wie mit anderen Fahrzeugen oder Zweiradfahrern. Während auf dem Stadtring klare Begrenzungen des Fahrbahnbereichs vorhanden sind, ist dies auf dem Parkplatz nicht der Fall. Der fahrbare Bereich definiert sich hier durch Flächen mit überfahrbaren Höhenunterschieden sowie Objektfreiheit.

3.3 Funktionsdefinition

Die Realisierung des Projektszenarios mit dem Versuchsträger umfasste eine Reihe von Einzelfunktionen (siehe Tabelle 3.2). Alle Einzelfunktionen gemeinsam bilden das Gesamtszenario ab, während Untergruppen jeweils die im Szenario vom Oktober 2010 bzw. den derzeitigen Stand der Implementierung abbilden.

Übergabe der Kontrolle: Um das Fahrzeug in den automatischen Modus zu versetzen, ist ein Übergabeprotokoll vorgesehen. Bei diesem wird zunächst der automatische Modus per Schlüsselschalter in Bereitschaft versetzt. Anschließend betätigt der Sicherheitsfahrer einen Knopf am Lenkrad und übergibt damit die Kontrolle an den Versuchsträger. Voraussetzung zur Übergabe ist die Anwesenheit des Versuchsträgers auf der vorgesehenen Strecke.

Funktion	Szenario Oktober 2010	Szenario Herbst 2012	Ziel-szenario
Übergabe der Kontrolle	✓	✓	✓
Ausparken vom Seitenstreifen	✗	✗	✓
Fahrstreifenwechsel	✗	✓	✓
Abbiegen bei bevorrechtigtem Verkehr	✗	✗	✓
Einparken in Parkboxen	✗	✗	✓
Wahrnehmung eines freien Parkplatzes	✗	✗	✓
Navigation auf einem Parkplatz	✗	✗	✓
Routenplanung auf dem Stadtring	✗	✓	✓
Folgen eines anderen Verkehrsteilnehmers	✓	✓	✓
Erfassung des Signalbildes von Lichtsignalanlagen (manuell)	✓	✓	✓
Erfassung des Signalbildes von Lichtsignalanlagen (automatisch)	✗	✓	✓
Navigation auf dem Stadtring	✓	✓	✓
Notbremsmanöver	✓	✓	✓
Wendemanöver	✓	✓	✓
Abbruch der automatischen Funktion	✓	✓	✓

Tabelle 3.2: Übersicht über für die Szenarien notwendigen Einzelfunktionen (Stand Herbst 2012)

Ausparken vom Seitenstreifen: Das Zielszenario beginnt auf dem Seitenstreifen zwischen anderen parkenden PKW. Aus dieser Position heraus soll das Fahrzeug in den fließenden Verkehr einfahren, ohne andere Teilnehmer zu behindern. Dabei muss es sowohl auf den rückwärtigen Verkehr als auch auf die Fahrzeuge vor, hinter und neben ihm achten und eine Bahn aus der Parklücke planen.

Fahrstreifenwechsel: Während der Fahrt auf dem Stadtring sind Fahrstreifenwechsel notwendig, um einerseits Hindernissen (z. B. langsameren Fahrzeugen oder statischen Objekten) auszuweichen und andererseits der Route zu folgen. Dabei erkennt das Fahrzeug zunächst eine freie Lücke, plant anschließend eine Trajektorie hinein und fährt diese ab. Dabei gilt es, sowohl auf den umliegenden Verkehr zu achten, um Kollisionen und Behinderungen zu vermeiden, als auch die Fahrstreifengeometrie und die daraus resultierenden Verkehrsregeln beachten.

Abbiegen bei bevorrechtigtem Verkehr: Die meisten Lichtsignalanlagen auf dem Stadtring besitzen ein gesondertes Signal für Linksabbieger. An den Stellen, an denen dies nicht erfüllt ist, muss der Versuchsträger eine geeignete Position auf der Kreuzung einnehmen, um den bevorrechtigten Verkehr wahrzunehmen und bei einer entsprechenden Lücke die Fahrt fortzusetzen. Hierbei ist zunächst der notwendige Bereich zur Überwachung zu bestimmen und zu überwachen. Verdeckungen des Erfassungsbereichs durch andere Objekte müssen dabei berücksichtigt werden.

Einparken in Parkboxen: Der Parkplatz am Ende der Route stellt Parkplätze als Parkbox zur Verfügung. Der Versuchsträger plant eine Route in eine ausgewählte Parkbox, ohne dabei mit den Nachbarfahrzeugen oder anderen Hindernissen zu kollidieren. Die Parkposition des Fahrzeugs muss ein komfortables Verlassen des Versuchsträgers ermöglichen.

Wahrnehmung eines freien Parkplatzes: Während der Navigation auf dem Parkplatz überwacht der Versuchsträger die Umgebung und vermisst mögliche Parklücken. Ist eine ausreichend große Parklücke sowie ausreichend Platz zur Anfahrt vorhanden, wird das Parkmanöver eingeleitet.

Navigation auf einem Parkplatz: Da auf einem Parkplatz i. d. R. keine Fahrstreifen vorhanden sind, muss das Fahrzeug die unmittelbare Umgebung auf ihre Befahrbarkeit hin bewerten. Dies kann z. B. durch die Messung der relativen Steigung sowie die Wahrnehmung von Objektfreiheit einer Fläche geschehen. Anhand dieser Befahrbarkeitskarte kann ein Pfad über den Parkplatz und ggf. auch zurück auf den Stadtring geplant werden, der durch das Fahrzeug kollisionsfrei befahren werden kann.

Routenplanung auf dem Stadtring: Bei Eingabe eines Ziels in der im Fahrzeug vorhanden Straßenkarte plant der Versuchsträger eine entsprechende Route. Dabei werden notwendige Fahrstreifenwechsel sowie die zurückgelegte Strecke berücksichtigt.

Folgen eines anderen Verkehrsteilnehmers: Oft ist eine Bewegung mit der maximal zulässigen Geschwindigkeit auf dem Stadtring nicht möglich. In diesem Fall muss die eigene Geschwindigkeit dem vorderen Fahrzeug angepasst und ein üblicher Sicherheitsabstand eingehalten werden. Um diese Funktion zu realisieren, ist zunächst die Bestimmung der Fahrstreifenzugehörigkeit der umliegenden Fahrzeuge notwendig. Das Fahrzeug im gleichen Fahrstreifen vor dem Versuchsträger legt dann die eigene Geschwindigkeit fest. Der Versuchsträger verringert ggf. seine Geschwindigkeit, bis er die Geschwindigkeit des vorausfahrenden Fahrzeugs erreicht hat.

Erfassung des Signalbildes von Lichtsignalanlagen (manuell): Die Verkehrsregelung an Kreuzungen geschieht auf dem Stadtring über Lichtsignalanlagen. Einige dieser Anlagen sind mit Sendern ausgestattet und senden ihren Zustand an den Versuchsträger. Die übrigen Lichtsignalanlagen kann das Fahrzeug nicht selbst wahrnehmen. Hierzu wird es mit einer Mensch-Maschine-Schnittstelle ausgestattet die es dem Beifahrer ermöglicht, das aktuelle Signalbild der Lichtsignalanlage einzugeben.

Erfassung des Signalbildes von Lichtsignalanlagen (automatisch): Neben der manuellen Eingabe des Signalbilds der Lichtsignalanlagen kann das Fahrzeug das Signalbild via Infrastructure2Car empfangen. Das Signalbild, das von den Lichtsignalanlagen empfangenen wird, ist bevorrechtigt gegenüber dem manuell eingegebenen.

Navigation auf dem Stadtring: Neben der Routenplanung ist zur Fahrt auf dem Stadtring eine detailliertere Planung sowie die Regelung der Fahrzeugaktorik notwendig. Dies umfasst die Planung der genauen Trajektorie und Geschwindigkeitsplanung entsprechend der Verkehrsregeln, des umgebenen Verkehrs und des Fahrtziels. Der Versuchsträger darf dabei nie schneller fahren als die zulässige Höchstgeschwindigkeit des aktuellen

Streckenabschnitts. Da sich im Fahrzeug mindestens zwei Passagiere (Fahrer und Beifahrer) zur Bedienung des Versuchsträgers befinden, ist auf eine komfortable Fahrweise zu achten. Neben der Längsführung übernimmt das Fahrzeug auch die Querführung. Dabei folgt es dem Fahrstreifen oder im Bereich von Kreuzungen dem berechneten Kurs.

Notbremsmanöver: Starke Verzögerungen von anderen Verkehrsteilnehmern und plötzlich auftretende Objekte können ein Notbremsmanöver des Versuchsträgers zur Verhinderung von Kollisionen notwendig machen.

Wendemanöver: Zur Realisierung der Szenarien Oktober 2010 und Herbst 2012 sind Wendemanöver (siehe Abbildung 3.1, Punkt 5 und 6) notwendig. Dabei wird eine Trajektorie auf die Gegenfahrbahn geplant und diese ausgeführt, während dabei der umliegende Verkehr überwacht und ggf. darauf reagiert wird.

Abbruch der automatischen Funktion: Im Falle einer Fehlfunktion oder eines unerwarteten Verhaltens des Versuchsträgers kann der Fahrer den automatischen Modus sofort beenden und selbst die Steuerung übernehmen. Dies kann durch einen Notausschalter, einen Eingriff an der Lenkung, dem Gas- oder Bremspedal oder über diverse andere Wege geschehen (Nothdurft u. a., 2011a).

3.4 Versuchsträger

Abbildung 3.2: Versuchsträger Leonie des Projekts Stadtpilot

Der Versuchsträger Leonie des Projekts Stadtpilot ist ein Volkswagen Passat Variant (siehe Abbildung 3.2). Das Fahrzeug wurde für den Betrieb als automatisches Versuchsfahrzeug durch die Volkswagen Forschung umgebaut. Auf diese Weise sind die Aktuatoren des Fahrzeugs über einen CAN-Bus ansteuerbar. Darüber hinaus ist das Fahrzeug mit einer zweiten Lichtmaschine ausgestattet, um ausreichend Energie bereitzustellen.

Leonie ist ferner mit insgesamt sechs Computern ausgestattet. Vier Computer sind für die Verarbeitung der Messdaten und die Umsetzung der Fahraufgabe vorgesehen. Jeder Rechner ist mit einem Intel Core 2 Quad Prozessor mit 2,6 GHz und 4 GB Speicher ausgestattet. Ein weiterer Rechner ist für die Bedienung und die Visualisierung von Messdaten vorgesehen. Er kann vom Beifahrersitz aus über einen Touchscreen bedient werden. Der sechste Rechner wird für Überwachungsaufgaben genutzt. Er setzt u. a. die Aufzeichnung aller im Fahrzeug anfallenden Messdaten um. Die Systeme sind über ein redundantes Ethernet-Netzwerk miteinander verbunden und verfügen darüber hinaus über die notwendige Menge an CAN-Bus-Schnittstellen zur Auswertung der Sensoren und zur Ansteuerung des Fahrzeugs.

3.5 Sensoren des Versuchsträgers

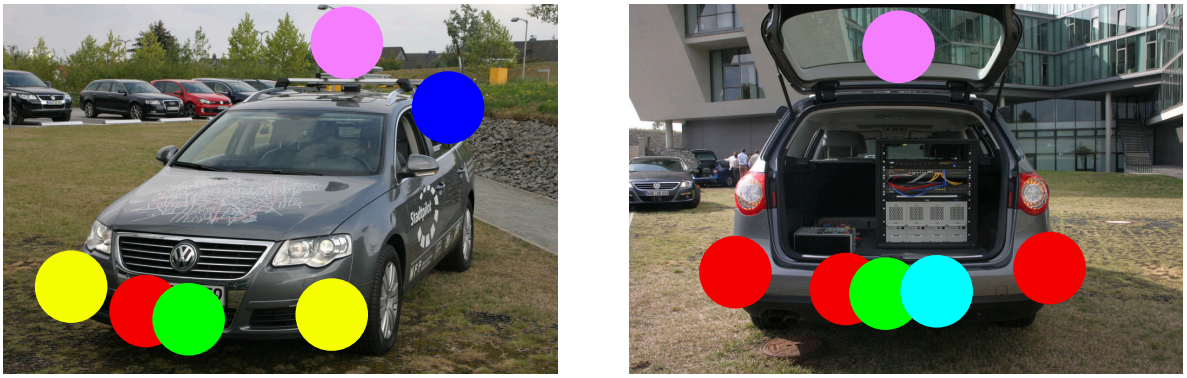


Abbildung 3.3: Sensoren am Versuchsträger Leonie. Gelb: IBEO Alaska XT, Cyan: IBEO Lux, Blau: Sick LMS100, Rot: SMS UMMR Radar, Grün: Hella IDIS 1/2, Violett: Velodyne HDL64ES2

Die am Versuchsträger angebrachten Sensoren sind in Abbildung 3.3 zu sehen. Im vorderen Stoßfänger sind links und rechts Sensoren des Typs IBEO Alaska XT montiert. In der Mitte findet sich ein SMS UMMR 2010-Radarsensor, flankiert von einem Hella IDIS und Hella IDIS 2-Sensor. Auf dem Dach ist ein Velodyne HDL64ES2 montiert. An der linken und rechten Seite des Fahrzeugs ist je ein Sick LMS100-Laserscanner zur Fahrstreifenmarkierungsdetektion angebracht. Am hinteren Stoßfänger des Fahrzeugs befindet sich an den Außenpositionen je ein SMS UMMR 2006 Blindspot-Radarsensor. In der Mitte des hinteren Stoßfängers befindet sich ein SMS UMMR 2006-Radar, ein weiterer Hella IDIS-Sensor sowie ein IBEO Lux-Laserscanner. Zur Ortung verfügt der Versuchsträger über ein DGPS/INS-System des Herstellers iMAR. Das System iTrace F200s ermöglicht es dem Fahrzeug, sich mit einer Genauigkeit von bis zu $2cm$ zu positionieren. Dazu wird das System mit Daten des HEPS-Diensts des Dienstleisters SAPOS versorgt, welche via NTRIP über UMTS empfangen werden.

Die die im Stadtpilot-Projekt verwendeten Umfeld wahrnehmenden Sensoren werden im Folgenden kurz vorgestellt.

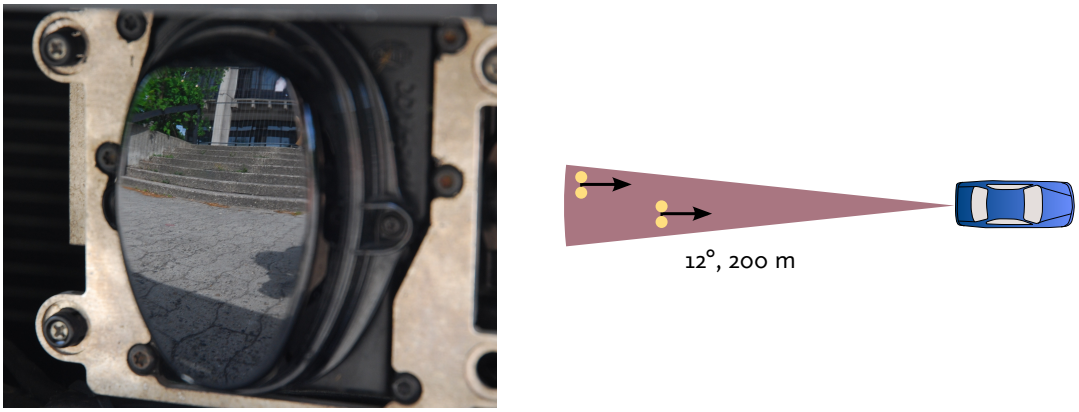


Abbildung 3.4: Hella IDIS-Sensor mit Erfassungsbereich und Objekthypothesen

Hella IDIS

Der IDIS-Sensor des Herstellers Hella ist ein Multi-Beam-Lidarsensor (Hella KGaA Hueck & Co, 2006). Abbildung 3.4 zeigt den Sensor an dem Versuchsträger des Projekts Stadtpilot. Der Hella IDIS enthält in der verwendeten Ausführung 12 einzelne Lasereinheiten, die in einem Winkel von 1° zueinander angeordnet sind. Durch den Sensor wird ein Erfassungsbereich von 12° mit einer Reichweite von bis zu $200m$ abgedeckt. Der Messfehler des Sensors ist mit $\pm(1\% + 1m)$ in lateraler Richtung und 1° in longitudinaler Richtung angegeben. Innerhalb des Sensors wird bereits eine Vorverarbeitung der Messrohdaten sowie ein Tracking und ggf. auch eine Fahrzeuglängsregelung durchgeführt. Das Tracking verfolgt maximal 25 Objekthypothesen. Jede der Objekthypothesen beschreibt den wahrgenommenen linken und rechten Rand des Objekts. Die darüber hinaus bereitgestellten Informationen, wie z. B. eine geschätzte Geschwindigkeitsinformation oder die Fahrstreifeninformationen, werden im Rahmen dieser Arbeit nicht genutzt, da sie sehr spezifisch auf den Einsatz als ACC-Sensor ausgerichtet sind.

Hella IDIS 2

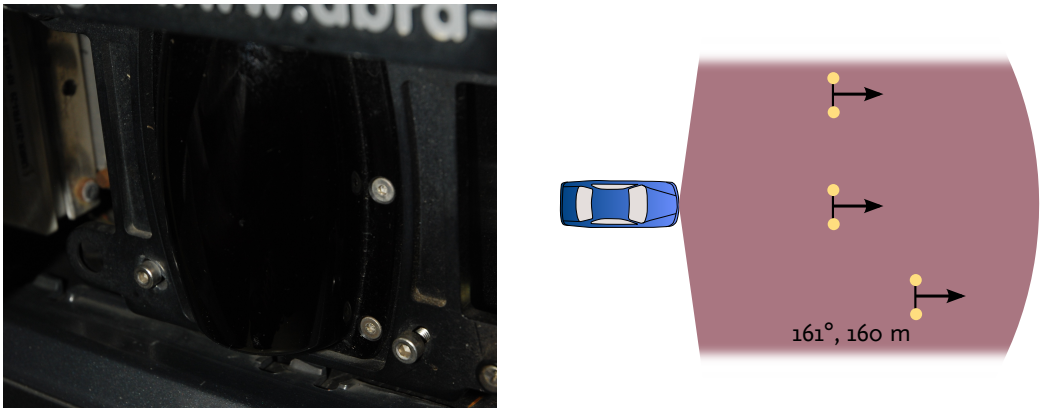


Abbildung 3.5: Hella IDIS 2-Sensor mit Erfassungsbereich und Objekthypothesen

Die Weiterentwickelung des Hella IDIS-Sensors ist der Hella IDIS 2-Sensor (Hella KGaA Hueck & Co, 2008). Er ist ein Vertreter der scannenden Lidarsensoren. Die im Projekt Stadtpilot verwendete Version des Sensors deckt einen Erfassungsbereich von $161°$ bei einer Auflösung von $1,8°$ ab. Durch den Sensor werden Objekte bis in eine Entfernung von $150m$ mit einem Messfehler von $\pm(1\% + 1m)$ wahrgenommen. Der Sensor führt intern eine Objekthypothesenbildung und ein auf eine ACC-Applikation ausgerichtetes Tracking durch. Jede Objekthypothese beschreibt den linken und rechten Rand eines Objekts sowie deren durch das Tracking abgeleitete Geschwindigkeit. Die Daten des Sensors werden innerhalb des Projekts Stadtpilot zur Realisierung von Folgefahrtsituationen genutzt.

IBEO Alaska XT

Ein weiterer Vertreter der scannenden Lidarsensoren ist der Alaska XT-Sensor des Herstellers IBEO (Ibeo Automobile Sensor GmbH, 2006) (siehe Abbildung 3.6). Der Scanner arbeitet mit einem Erfassungsbereich von $\pm 164°$ bis zu einer Entfernung von $200m$. Die Winkelauflösung des Sensors ist abhängig von der aktuellen Blickrichtung des Laserstrahls und bewegt sich zwischen $2°$ und $0,125°$. Im Gegensatz zu den Lidarsensoren der IDIS-Reihe besitzt dieser Scanner vier Laserebenen. Diese sind zwischen $\pm 1,6°$ zur Hauptachse angeordnet und ermöglichen es, eine seicht ansteigende Fläche von einer senkrechten Wand zu unterscheiden. Vorwiegend wird diese Fähigkeit zur Detektion von Reflexionen an der Fahrbahnfläche genutzt. Neben der Mehrebenentechnologie bietet der Sensor auch die Fähigkeit, mehr als eine Reflexion pro Laserstrahl wahrzunehmen. Dies tritt in dem Fall auf, dass ein Laserstrahl ein Objekt nicht exakt trifft, sondern es lediglich streift. Auf diese Weise reflektieren auch dahinter stehende Objekte den Strahl.

Mehrere dieser Sensoren können über ein Steuergerät zu einer Fusionseinheit zusammengeschlossen werden. Mithilfe des Steuergeräts kann eine Vorverarbeitung der Messdaten, eine Objekthypothesenbildung sowie ein Tracking dieser Hypothesen durchgeführt werden. Das Ergebnis des Trackings ist eine Objekthypothese mit offenem Polygonzug und einem Geschwindigkeitsvektor im 2D-Raum. Darüber hinaus ist auch der Empfang einer 2D-Messpunktwolke möglich. Die Anbindung des Sensors erfolgt über ein ARCnet-Netzwerk. Eine Anbindung über den CAN-Bus wird nur über die externe Fusionseinheit ermöglicht. Darüber hinaus ist es möglich, die CAN-Botschaften sowie die Messpunktpunkte über Ethernet zu empfangen.

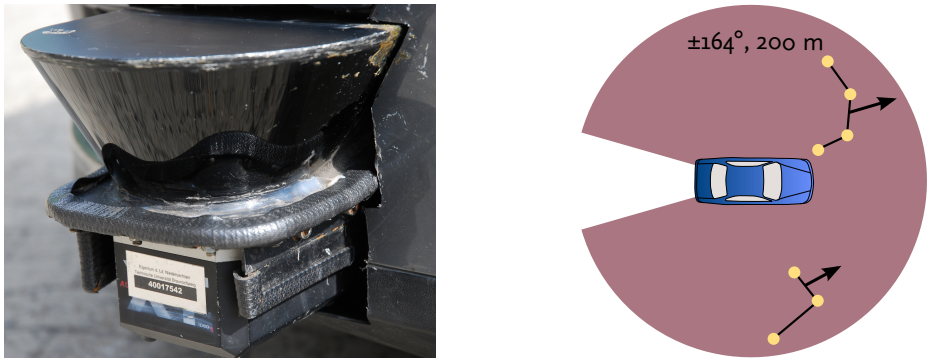


Abbildung 3.6: IBEO Alaska XT-Sensor mit Erfassungsbereich und Objekthypothesen

IBEO Lux

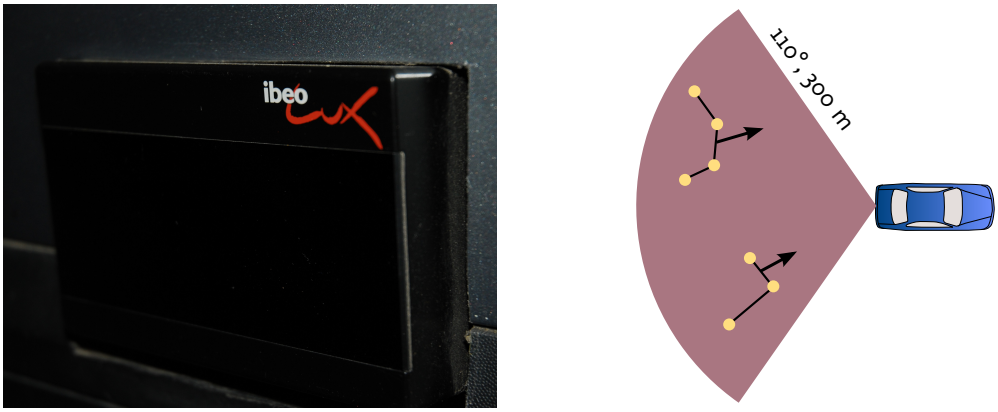


Abbildung 3.7: IBEO Lux-Sensor mit Erfassungsbereich und Objekthypothesen

Der Nachfolgesensor des IBEO Alaska XT ist der IBEO Lux (Ibeo Automobile Sensor GmbH, 2008) (siehe Abbildung 3.7). Der Sensor wurde auf Grundlage des Vorgängersystems entwickelt. Dabei wurde der Schritt von der Einzel- zur Kleinserienproduktion vollzogen. Hierzu hat sich das Gehäuse erheblich geändert und das Steuergerät wurde mit in den Sensor integriert. Der Erfassungsbereich des Sensors beträgt nur noch $110°$ mit einer maximalen Entfernung von ca. $300m$ bei einer vertikalen Auflösung von den bereits vom IBEO Alaska XT bekannten $\pm 1,6°$. Während beim IBEO Alaska XT die vier Laserebenen aufgrund der mechanischen Konstruktion nur entlang der Hauptachse des Sensors vollständig aufgefächert sind, ist beim IBEO Lux-Sensor der Bereich durch mechanische Veränderungen auf von $35°$ bis $-45°$ ausgedehnt. Hierdurch kann der Sensor diese Technologie noch besser nutzen und Bodenreflexionen wahrnehmen.

Ein weiterer Schwerpunkt bei der Entwicklung des IBEO Lux waren die verfügbaren Schnittstellen. Gegenüber dem Vorgängermodell wurden die Protokolle überarbeitet und vereinfacht. Der Sensor stellt seine Daten nun direkt über eine CAN- oder Ethernet-Schnittstelle bereit.

Velodyne HDL64ES2

Während die bisher vorgestellten Sensoren sich in ein Fahrzeug integrieren lassen, ist der Velodyne HDL64ES2-Laserscanner dafür ungeeignet (Velodyne Lidar, Inc, 2008) (siehe Abbildung 3.8). Er deckt mit seinem rotierenden Lasermodul einen Bereich von $360°$ um das Fahrzeug bis zu einer Entfernung von ca. $100m$ ab. Die Stärke des Sensors liegt in der hohen Dichte von Messpunkten. Insgesamt stehen 64 Laserebenen in einem Bereich von $-24,8°$ bis $2°$ zur Hauptachse zur Verfügung, die zyklisch die Umgebung abtasten. Diese bis zu $2,3 \cdot 10^6$ Messpunktpunkte werden über eine Ethernet-Schnittstelle bereitgestellt. Eine Vorverarbeitung über die Signalauswertung hinaus oder Objekthypothesenbildung wird durch den Sensor nicht durchgeführt.

Abbildung 3.8: Velodyne HDL64ES2 mit Erfassungsbereich und Messpunktwolke einzelner Laserebenen

SMS UMMR aus dem Jahr 2006

Der Radarsensor UMMR des Herstellers smart microwave Sensors GmbH in der Version des Jahres 2006 kann, je nach Modulationsverfahren und Antennentyp, für unterschiedliche Aufgaben eingesetzt werden (smart microwave sensors GmbH, 2006) (siehe Abbildung 3.9). Der Sensor lässt sich sowohl als Dauerstrich- als auch im Pulsverfahren im 24 GHz Bereich betreiben. In Abhängigkeit des Antennentyps lässt er sich beispielsweise im FMCW-Modus für ACC-Anwendungen nutzen. In diesem Fall wird die Reichweite mit $160m$ (Fahrzeug) und einem Öffnungswinkel von $\pm 20°$ angegeben. Zur Überwachung des toten Winkels eines Fahrzeugs kann das Radar mit einem anderen Antennentyp im Nahbereich eingesetzt werden. Hierbei wird ein Bereich von $\pm 70°$ und eine Reichweite von bis zu $10m$ erreicht.

Die Genauigkeit der Entfernungsmessung wird im FMCW-Modus mit $\pm 0,5m$ bis zu einer Entfernung von $10m$ angegeben. Darüber hinaus beträgt die Genauigkeit der Entfernung $\pm 5\%$. Der Geschwindigkeitswert wird durch den Sensor mit einer Präzision von $\pm 0,25\frac{km}{h}$ bestimmt. Im Pulsverfahren wird die Entfernung mit einer Genauigkeit von $\pm 13,5cm$ bestimmt. Das Geschwindigkeitssignal unterliegt dabei einem Messfehler von $\pm 0,8\frac{km}{h}$.

Der Sensor stellt die Ergebnisse der Messungen über ein CAN-Interface zur Verfügung.

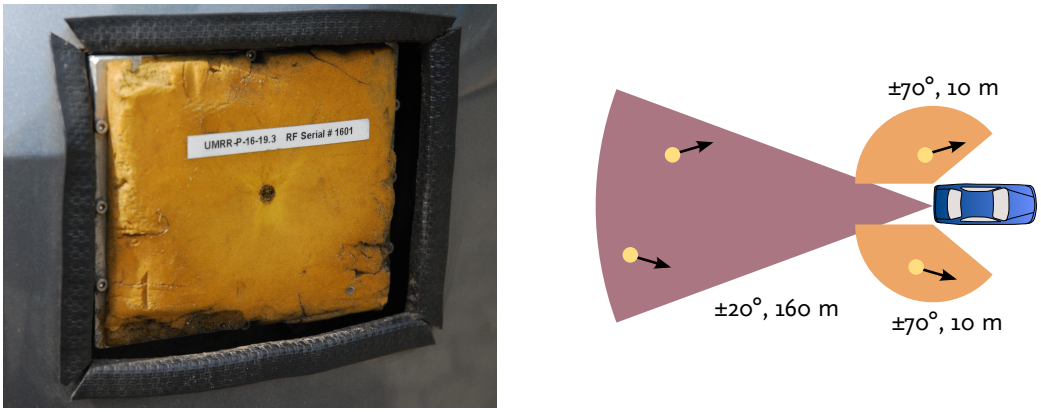


Abbildung 3.9: SMS UMMR-Sensor Jahr 2006 mit Antennentyp 19.3, Erfassungsbereich für Antennentypen 14/19.3 und Objekthypothesen

Dabei führt er eine Objekthypothesenbildung, aber kein Tracking durch. Mehrere Sensoren lassen sich über ein gemeinsames Steuergerät zu einer Einheit kombinieren. Das Steuergerät fusioniert die Daten miteinander und führt auf Wunsch auch ein Tracking durch.

SMS UMMR aus dem Jahr 2010

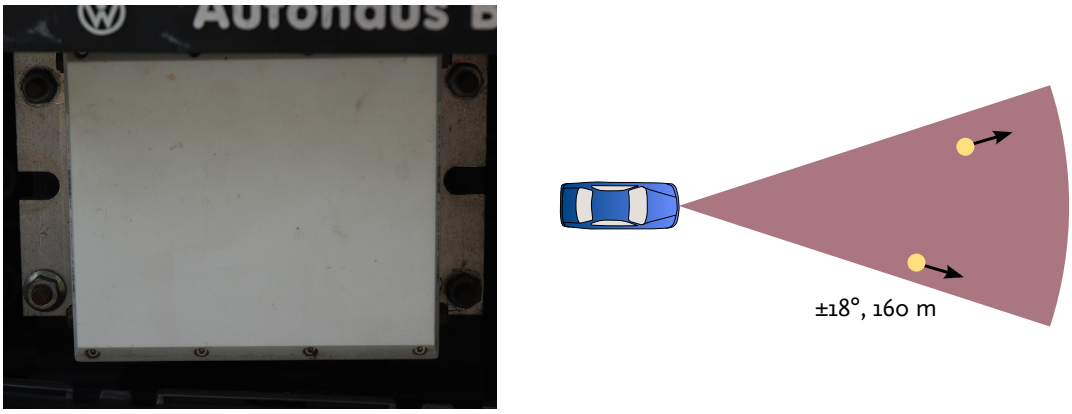


Abbildung 3.10: SMS UMMR-Sensor Jahr 2010 mit Antennentyp 29, Erfassungsbereich und Objekthypothesen

Gegenüber der Version aus dem Jahr 2006 wurde der Sensor in der Generation von 2010 weiterentwickelt (smart microwave sensors GmbH, 2010) (siehe Abbildung 3.10). Die Antennentypen wurden ergänzt und verändert. Für den vorliegenden Sensor sind drei Antennentypen dokumentiert. Die für ACC-Anwendungen vorgesehenen Antennentypen decken einen Bereich von $\pm 18°$ bzw. $\pm 35°$ bei einer Reichweite von $160m$ (Fahrzeug) bzw. $90m$ (Fahrzeug) ab. Eine Totwinkelüberwachung lässt sich nicht mehr umsetzen. Dafür ist nun ein Antennentyp für Spurwechselassistenten verfügbar. Der Typ 31 deckt dabei einen Bereich von $\pm 60°$ bei einer Reichweite von bis zu $60m$ ab. Die laterale Genauigkeit im FMCW-Modus beträgt im Bereich bis $10m \pm 0,25m$ und wird ab $10m$ mit $\pm 2,5\%$ angegeben. Der Geschwindigkeitsvektor wird mit einer Genauigkeit von $\pm 0,25 \frac{km}{h}$ bereitgestellt. Ein Pulsverfahren bietet der Sensor nicht mehr.

3.6 System- und Softwarearchitektur

Alle Computer des Versuchsträgers Leonie laufen unter den Betriebssystem Linux. Der Betriebssystemkern weist eine ausreichende Zuverlässigkeit und Reaktionszeit auf (Erfahrung der Entwickler) und macht es somit möglich, die Anforderungen der Fahrzeugregelung und sowie die Zyklusstabilität von Algorithmen zu erfüllen. Die sechs installierten Rechner werden über den Dienst NTP auf Basis der Zeitinformation des GPS/INS-Systems zeitlich synchronisiert. Durch die Verfügbarkeit einer gemeinsamen Zeitbasis wird die Kommunikation zwischen den einzelnen Modulen erheblich vereinfacht und kommunikations- und rechenzeitbedingte Latenzen können in den Algorithmen berücksichtigt werden. Als Kommunikationsframework wird der Data Distribution Service des Herstellers RTI genutzt (Real-Time Innovations Inc., 2011). Dieser Service basiert auf dem Publisher-Subscriber-Architekturmuster und ermöglicht es einzelnen Softwaremodulen Daten bereitzustellen, ohne

zu wissen, wer die Abnehmer sind und ob es welche gibt. Kommunikationswege können sich somit zur Laufzeit reorganisieren oder auch redundante Datenquellen bereitgestellt werden. Als Applikationsframework kommt das Automotive Data and Time-triggered Framework (ADTF) zum Einsatz (Audi Electronic Venture GmbH, 2011). Dieses Framework basiert auf dem Pipes-and-Filters-Architekturmuster und eignet sich damit hervorragend zur Verarbeitung von Datenströmen. Innerhalb des ADTFs werden einzelne Komponenten als sogenannte Filter umgesetzt. Diese definieren sich nur über eine Menge von typisierten Ein- und Ausgängen und bieten somit die Möglichkeit der einfachen Wiederverwendung in anderen Kontexten oder Projekten. Ergänzt wird das ADTF von diversen Toolboxen (z. B. zur Visualisierung oder Hardwareansteuerung).

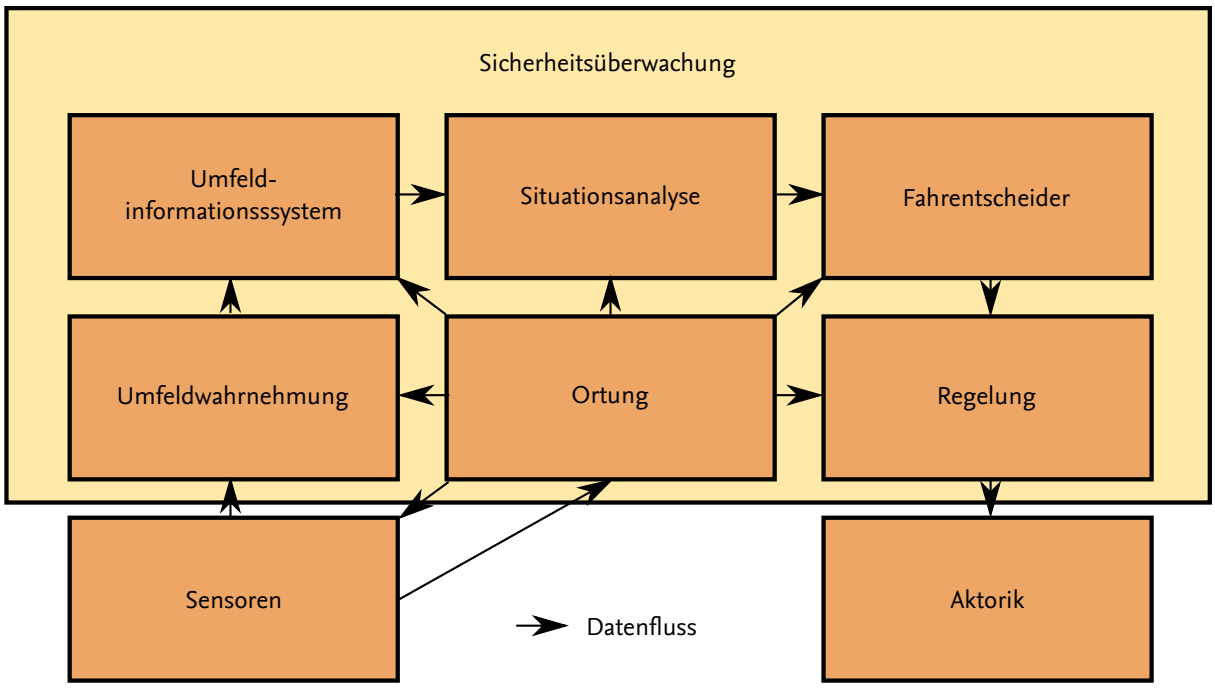


Abbildung 3.11: Softwaremodule des Projekts Stadtpilot mit Datenfluss

Abbildung 3.11 zeigt eine abstrakte Darstellung der im Projekt Stadtpilot eingesetzten Softwaremodule. Jedes dieser Module besteht aus einer Vielzahl von kleineren Komponenten, auf die hier nicht näher eingegangen wird. Im Folgenden wird jedoch ein kurzer Überblick über die Funktionsweise der einzelnen Module als Ganzes gegeben:

Sensoren: Das Sensorik-Modul empfängt die Datenströme der einzelnen Sensoren und stimuliert die Sensoren wiederum mit den zum Betrieb notwendigen Informationen. Dazu wurden für jeden Sensor die in Abschnitt 9.1 beschriebenen Klassen implementiert.

Umfeldwahrnehmung: Im Modul Umfeldwahrnehmung werden die durch die Sensoren gemessenen Daten verarbeitet und zu einem gemeinsamen konsistenten Bild des aktuellen Fahrzeugumfelds fusioniert. Das Modul wird in den folgenden Abschnitten detailliert behandelt. Die Ableitung von konkreten Anforderungen an die Umfeldwahrnehmung und die Beschreibung der Algorithmen erfolgt in Teil III.

Ortung: Durch die hohe Verkehrsdichte im innerstädtischen Bereich und die hohe Anzahl an Infrastrukturelementen ist eine genaue Ortung unabdingbar. Hierzu wird im Versuchsträger Leonie ein GPS/INS-System eingesetzt, welches mithilfe von Korrekturdaten eine Positionsgenauigkeit von bis zu $2cm$ erreichen kann. Durch Reflexionen und Abschattungen durch Gebäude ist dies jedoch nicht immer möglich. Aus diesem Grund wird die Ortung durch Lasersensoren gestützt werden. Diese detektieren die Fahrbahnmarkierungen und vergleichen sie mit den gespeicherten Positionen in einer digitalen Karte. Auf diese Weise kann eine ausreichende Ortung auch in Situationen ohne ausreichendes GPS-Signal gewährleistet werden.

Umfeldinformationssystem: Das Umfeldinformationssystem des Versuchsträgers kombiniert das aktuelle Fahrzeugumfeld mit Informationen aus einer digitalen Karte (Nothdurft u. a., 2011a). Diese Karte enthält den Strecken- und Fahrstreifenverlauf genauso wie ortsgebundene Verkehrsregeln und die Position von Lichtsignalanlagen. Darüber hinaus bietet dieses Modul eine Reihe von Dienstleistungen für die Situationsanalyse an. Beispielsweise kann das Modul Objekthypothesen einzelnen Fahrstreifen zuordnen oder Objekthypothesen unter Verwendung der Informationen der digitalen Karte über größere Zeiträume prädizieren.

Situationsanalyse: In enger Zusammenarbeit mit dem Umfeldinformationssystem untersucht die Situationsanalyse die Daten des aktuellen Fahrzeugumfelds (Ulbrich, 2011). Hierzu werden bestimmte Manöver (z. B. Fahrstreifenwechsel) unter der Berücksichtigung der Daten des Umfeldinformationssystems bewertet.

Fahrentscheider: Welche Aktion letztlich angewendet wird, wertet der Fahrentscheider aus (Ulbrich, 2011). Er trifft unter Berücksichtigung der aktuellen Missionsziele eine Entscheidung über das nächste Fahrmanöver. Darüber hinaus wird eine Trajektorie geplant, welche anschließend durch die Regelung umgesetzt wird (Wille u. a., 2010).

Regelung: Die geplante Trajektorie und das geplante Geschwindigkeitsprofil des Fahrentscheiders werden durch die Regelung umgesetzt. Hierzu werden die notwendigen CAN-Botschaften erzeugt, um Gas, Bremse und Lenkung des Fahrzeugs anzusteuern.

4 Zusammenfassung

Der vergangene Abschnitt motivierte zunächst die Arbeit und gab einen Einblick über den Kontext, in dem sie sich bewegt. Daran anschließend wurde in Abschnitt 2.1 auf unterschiedliche Projekte zum automatischen Fahren im innerstädtischen Bereich während des Zeitraums des Projekts Stadtpilot eingegangen.

Die Anwendung der in dieser Arbeit beschriebenen Architektur liegt in der Fusion von Umfeld wahrnehmenden Sensoren. Im Rahmen von Abschnitt 2.2 wurden unterschiedliche Sensortechnologien vorgestellt. Eines der Hauptbestandteile von Fusionssystemen sind Filteralgorithmen. Die Abschnitte 2.3 und 2.4 gaben eine kurze Einführung in die mathematischen Grundlagen von Filteralgorithmen für objekthypothesen- und gitterbasierte Fusionssysteme.

Der nun folgende Teil der Arbeit beschäftigt sich ausführlich mit der Beschreibung einer Softwarearchitektur zur Umfeldwahrnehmung. In Abschnitt 2.5 wurden dazu bestehende Architekturen vorgestellt und ihre Vor- und Nachteile in Bezug auf die in Teil II beschriebene Architektur eingegangen.

Die vorliegende Arbeit wurde im Rahmen des Projekts Stadtpilot angefertigt. In Abschnitt 3 wurde dieses Projekt ausführlich vorgestellt. Die Projektziele wurden erläutert, das Fahrzeugumfeld näher beschrieben sowie die Gesamtaufgabe in einzelne Funktionsanforderungen aufgespalten. Anschließend wurde auf den Versuchsträger mit seinen Sensoren sowie die System- und Softwarearchitektur eingegangen. Eine Ableitung von Anforderungen an die Umfeldwahrnehmung erfolgt in Teil III.

Teil II: Softwarearchitektur der Fusion von Umfeld wahrnehmenden Sensoren

Der folgende Teil gibt zunächst eine allgemeine Einführung in die Thematik Softwarearchitektur. Darauf folgend wird eine domänenspezifische Softwarearchitektur zur Fusion von Umfeld wahrnehmenden Sensoren im Automobil vorgestellt. Diese Architektur, der Architekturkontext und die architekturspezifischen Anforderungen dienen als Grundlage für die weiteren Abschnitte. Anschließend wird die Architektur zur Fusion von Umfeld wahrnehmenden Sensoren aus der Baustein- und der Laufzeitsicht beschrieben. Abgeschlossen wird der Teil mit einer Bewertung der aufgestellten Anforderungen.

5 Softwarearchitektur

In der Vergangenheit wurden viele Softwareprojekte ohne detaillierte Planung der zugrunde liegenden Strukturen begonnen. Das Ergebnis dieses Vorgehens resultierte oft im Scheitern der Projekte oder in einem erheblich überzogenen Kosten- und Zeitrahmen. Aus diesem Grund rückte das Thema Architektur immer mehr in den Focus (Starke u. Hruschka, 2011, Seite 1). Gerade durch den erheblich gestiegenen Anteil an interdisziplinären Softwareprojekten, die früher ausschließlich dem Maschinenbau oder der Elektrotechnik zugeordnet worden wären, ist eine Abstimmung zwischen allen Projektbeteiligten immer wichtiger.

Softwarearchitekten stellen die Verbindung zwischen unterschiedlichen Projektbeteiligten dar. Sie moderieren Diskussionen, (er)klären Anforderungen und Randbedingungen, entwerfen Strukturen, treffen Technologieentscheidungen, entwerfen technische Konzepte und begleiten die Umsetzung der Projekte. Darüber hinaus sind sie für die Kommunikation der Architektur zu allen Projektbeteiligten sowie die Bewertung der Architektur zuständig (Starke u. Hruschka, 2011, Seite 9ff). Dieses breite Tätigkeitsfeld hat in den letzten Jahren zu einem eigenen Berufsbild, dem des Softwarearchitekten, in vielen größeren Firmen geführt. Allgemein gehen Starke u. Hruschka davon aus, dass pro 30 Projektmitarbeiter ein Softwarearchitekt ausschließlich mit der Organisation und dem Entwurf der Softwarearchitektur beschäftigt sein sollte.

5.1 Begriffsdefinition: Softwarearchitektur

Softwarearchitektur wird in der Literatur auf viele ähnliche Weisen definiert. Eine Quelle für unterschiedliche Definitionen ist die Liste der Software Architecture Definitions des Software Engineering Instituts der Carnegie Mellon University (Software Engineering Institute at Carnegie Mellon University). Sie hat klassische Definitionen, Definitionen aus Büchern oder Definitionen der Community gesammelt.

Die Norm IEEE Std 1471:2000 definiert Softwarearchitektur beispielsweise folgendermaßen:

> „Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.“ (NORM IEEE Std 1471, 2000, Seite 14)

Clements u. a. definieren Softwarearchitektur ähnlich:

> „The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both.“ (Clements u. a., 2010, Seite 494)

In Vogel u. a. (2008) wird Softwarearchitektur wie folgt definiert:

> „Die Softwarearchitektur eines Systems beschreibt dessen Software-Struktur respektive dessen -Strukturen, dessen Software-Bausteine sowie deren sichtbare Eigenschaften und Beziehungen zueinander.“ (Vogel u. a., 2008, Seite 48)

Im Vergleich untereinander unterscheiden sich die obigen drei Definitionen nicht wesentlich. Alle drei definieren Softwarearchitektur als die Beschreibung der grundlegenden Strukturen des Systems bestehend aus deren Komponenten mit ihren Eigenschaften und Beziehungen zueinander bzw. der Projektumgebung.

Im Folgenden wird für den Begriff der Softwarearchitektur der Definition nach Starke u. Hruschka (2011) gefolgt. Sie wird für die Vorlage zur Softwarearchitekturbeschreibung „arc42“ nach Hruschka u. Starke (2011) genutzt. Das Vorgehen im Folgenden ist an diese Vorlage angelehnt, wurde aber, soweit das für diese Arbeit sinnvoll erschien, angepasst. Die zugehörige Definition von Softwarearchitektur lautet:

> „Die Architektur eines Systems beschreibt die Strukturen des Systems, dessen Bausteine, Schnittstellen und deren Zusammenspiel. Die Architektur besteht aus Plänen und enthält meistens Vorschriften oder Hinweise, wie das System erstellt oder zusammengebaut werden sollte.“ (Starke u. Hruschka, 2011, Seite 2f)

Für eine Softwarearchitektur wurde von Kruchten die Darstellung derselben durch unterschiedliche Sichten vorgeschlagen. Im 4+1-Views-Modell wird die Architektur auf vier verschiedene Weisen betrachtet (Kruchten, 1995). Die Darstellung als Logical View, Process View, Development View, Physical View und Scenarios ermöglicht es für jeden Entwickler die Informationen zu beschreiben, die für seine Arbeit notwendig sind. Ferner wird damit die Herausforderung, alle Informationen auf einmal in der gleichen Notation darstellen zu müssen, gelöst. Da im Folgenden in Anlehnung an die arc42-Vorlage vorgegangen wird, werden die Inhalte mithilfe der dort genutzten Begriffe beschrieben. Die Bausteinsicht (Logical View) beschreibt den Zusammenhang von Klassen, Modulen und Komponenten zueinander. Die Laufzeitsicht (Prozess View) geht näher auf das Zusammenspiel einzelner Objekte ein. Diese Sicht beschreibt beispielsweise Arbeitsabläufe auf unterschiedlichen Abstraktionsgraden. Die Verteilungssicht (Physical View) beschreibt die Verteilung von Komponenten und Modulen auf unterschiedliche Systeme zur Laufzeit. Hierbei können u. a. auch die Themen Zuverlässigkeit, Datendurchsatz und Verfügbarkeit mit betrachtet werden. Der Development View ist nicht Teil der arc42-Vorlage und beschreibt die Zusammenarbeit der Projektmitarbeiter und die zugehörigen Prozesse. Die Scenarios (das „+1“) des 4+1-Modells beschreiben typische Anwendungsfälle des Projekts und gehen sowohl in der Laufzeit- als auch der Verteilungssicht auf.

In dieser Arbeit wird die Bausteinsicht in Abschnitt 9 behandelt. Die Laufzeitsicht folgt in Abschnitt 10. Auf die Verteilungssicht wird z. T. in Teil III und Abschnitt 3.6 eingegangen. Die zugrunde liegenden Szenarien sind einerseits in der Projektbeschreibung in Abschnitt 3 bzw. in den Anforderungen in den Abschnitten 7 und 13 und andererseits in der logischen Architektur einer Fusionsstruktur in Abschnitt 8 enthalten.

5.2 Verwendete Software- und Architekturmuster

Zur Erstellung von Software und von Architekturen haben sich Entwurfsmuster durchgesetzt. Diese wurden erstmals in Alexander u. a. (1977) beschrieben. In ihrem Buch gehen Alexander u. a. zwar auf Architekturmuster bei Gebäuden oder Städten ein, die zugehörige Definition entspricht jedoch der von Softwarearchitekturmustern und wird deshalb allgemein als Ursprung der Softwarearchitekturmuster betrachtet:

> „Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.“ (Alexander u. a., 1977, Seite X)

In der vorliegenden Arbeit werden verschiedene Entwurfsmuster wiederholt während des Entwurfs der Architektur und von Algorithmen angewendet. Eine Übersicht weiterer Muster kann in Johnson u. a. (1995) oder Gamma u. a. (2004) nachvollzogen werden. Die hauptsächlich genutzten Muster werden im Folgenden kurz erklärt:

Pipes-and-Filters: Im Pipes-and-Filters-Muster wird ein Datenstrom verarbeitet. Jeder Verarbeitungsschritt (Filter) greift dabei nur auf die eingehenden Daten zu und stellt die Ergebnisse über einen ausgehenden Datenpuffer (Pipe) bereit. Alle Filter werden über diese Datenpuffer miteinander verbunden und verarbeiten so den Datenstrom (Buschmann u. a., 1996, Seite 200ff).

Shared-Repository: Das Shared-Repository-Muster stellt eine Kommunikationsstrategie für Komponenten mit gemeinsamem Speicher dar. Innerhalb des Systems wird ein zentraler Speicher bereitgestellt. Jede Komponente kann Daten in den Speicher hinzufügen und in den Daten lesen. In der ursprünglichen Form stellt der zentrale Speicher die einzige Kommunikationsform zwischen den Komponenten dar (Lalanda, 1998).

Layer: Mit dem Layer-Muster wird eine horizontale Aufteilung eines Systems vorgenommen. Einzelne Aufgaben werden bestimmten Ebenen zugeordnet. Eine Ebene kann jeweils nur mit der darunter bzw. darüber liegenden Ebene über Schnittstellen kommunizieren (Buschmann u. a., 1996, Seite 185).

Fassade: Möchte man die internen Schnittstellen eines Systems verbergen, so können diese zu einer gemeinsamen (vereinfachten) Schnittstelle zusammengefasst werden. Diese Fassade wird nun von allen außenstehenden Elementen ausschließlich angesprochen (Gamma u. a., 2004, Seite 212ff).

Publisher-Subscriber: Durch das Publisher-Subscriber-Muster können Sender und Empfänger eines Datenstroms voneinander entkoppelt werden. Dabei übermittelt ein Sender (Publisher) über eine Infrastruktur das Angebot einen Datenstrom zu senden. Andere konsumierende Komponenten (Subscriber) können sich für diesen Datenstrom registrieren und erhalten so Zugriff auf die Daten (Buschmann u. a., 2007, Seite 234ff).

5.3 Anpassungen der Unified Modeling Language

Die Darstellungen dieses Teils folgen vorwiegend der Unified Modeling Language (UML) (Rupp u. a., 2005). Dabei werden vier Arten von Diagrammen eingesetzt. Aktivitätsdiagramme (Rupp u. a., 2005, Seite 265ff) finden Anwendung in Abschnitt 8 zur Beschreibung der logischen Architektur. Im Abschnitt 9 werden Komponentendiagramme (Rupp u. a., 2005, Seite 213ff) genutzt, um gröbere Strukturen der einzelnen Softwarebausteine zu beschreiben. Diese werden anschließend durch Klassendiagramme (Rupp u. a., 2005, Seite 95ff) präzisiert und detailliert. Der vierte Diagrammtyp stellt das Sequenzdiagramm (Rupp u. a., 2005, Seite 407ff) dar. Es wird in Abschnitt 10 einsetzt, um exemplarische Abläufe der Datenverarbeitung innerhalb und zwischen den in Abschnitt 9 beschriebenen Strukturelementen aufzuzeigen.

Zur Anpassung der Sprache an die Bedürfnisse eines Projekts können u. a. Stereotypes eingesetzt werden. Sie werden an dem dargestellten Element zwischen französischen Guillemets angezeigt (≪Stereotype≫). Folgende Stereotypes finden im Folgenden Anwendung:

Application Specific Datatype: Von den Anforderungen der verarbeitenden Applikation abhängiger Datentyp.

Drivability: Projektspezifischer Datentyp aus der Arbeit von Effertz (2008).

Enumeration: Datentyp, der nur eine bestimmte Menge an Zuständen annehmen kann.

HeightProfile: Projektspezifischer Datentyp aus der Arbeit von Effertz (2008).

Interface: Ein Interface is eine Art von Klassifikator, welche die Deklaration einer Menge von zusammenhängenden öffentlichen Eigenschaften und Verpflichungen repräsentiert. (Übersetzung von Rupp u. a. (2005, Seite 119) nach Object Management Group (2005, Seite 82))

Project Specific: Architekturelemente, die projektbezogen implementiert werden müssen. Sie stellen nur beispielhafte Repräsentationen dar.

SensorDataObjectDeEncoder: Realisierung des Architekturelements *SensorDataObjectDeEncoder*.

SensorDataPointCloudDeEncoder: Realisierung des Architekturelements *SensorDataPointCloudDeEncoder*.

UnifiedPointCloudPipe: Kommunikation über eine *UnifiedPointCloudPipe*.

UnifiedSensorObjectListPipe: Kommunikation über eine *UnifiedSensorObjectListPipe*.

Darüber hinaus werden wiederkehrende Elemente (z. B. Schnittstellen) nicht in allen Diagrammen vollständig wiederholt. Diese Elemente sind mit drei Punkten (. . .) gekennzeichnet.

6 Systemumfang der Softwarearchitektur zur Fusion von Umfeld wahrnehmenden Sensoren und Abgrenzung gegenüber dem Projektkontext

Mithilfe der im Folgenden vorgestellten Softwarearchitektur soll ein Umfeldwahrnehmungssystem im automotiven Bereich realisiert werden. Hierzu werden Daten von Umfeld wahrnehmenden Sensoren empfangen, verarbeitet und schließlich einer Applikation zur Verfügung gestellt. Im automotiven Bereich haben sich derzeitig zwei Sensordatenfusionsarten durchgesetzt: die objekthypothesen- und die gitterbasierte Fusion. Beide sollen im Rahmen dieser Arbeit durch eine Architektur abgebildet werden.

Die objekthypothesenbasierte Fusion verarbeitet eingehende Messdaten von einem oder mehreren Umfeld wahrnehmenden Sensoren und bildet diese auf Objekthypothesen ab. Eine Objekthypothese beinhaltet u. a. die Annahme über die Existenz eines Objekts in der realen Welt. Dieses Objekt wird durch einen Zustandsvektor beschrieben (z. B. Position und Geschwindigkeitsvektor). Die in die Fusion eingehenden Daten können dabei sowohl direkte Messdaten der Sensoren als auch bereits gebildete oder sogar getrackte Objekthypothesen sein.

Die gitterbasierte Fusion bildet einen oder mehrere Messwerte auf eine gitterbasierte Datenstruktur ab. Diese beschreibt im einfachsten Fall einen Ausschnitt der realen Welt durch die Kombination von zwei-dimensionalen Bereichen (Gitterzellen). Die Relation zwischen den Gitterzellen wird durch eine mathematische Vorschrift beschrieben (z. B. äquidistanter Abstand, radiale Anordnung). Jedes Gitter besitzt ein lokales diskretes Koordinatensystem mit der kleinsten Einheit „Gitterzelle“ und ist relativ zu einem anderen Koordinatensystem definiert.

Zusätzlich zur Systemumfang der Architektur ist die Abgrenzung gegenüber deren Umgebung sinnvoll und notwendig (Sommerville, 2004, Seite 171). In einem Kontextdiagramm werden hierzu die einzelnen Stakeholder des Systems bzw. die Stakeholder, die mit dem System interagieren, identifiziert und in Beziehung zueinander gesetzt. „Ein Stakeholder eines Systems ist [dabei] eine Person oder Organisation, die (direkt oder indirekt) Einfluss auf die Anforderungen des betrachteten Systems hat.“ (Pohl u. Rupp, 2011, Seite 12) Während der Identifikation der Stakeholder werden auch die Systemgrenzen der Architektur festgelegt. Diese Grenzen trennen das zu entwickelnde System von seiner Umgebung ab und definieren dadurch, welche Bestandteile während der Entwicklung ohne Rücksprache mit den Stakeholdern verändert werden können und welche nicht. Ferner ist es durch die frühzeitige Festlegung möglich, Randbedingungen besser abzuschätzen, Zuständigkeiten eindeutiger zu klären sowie die Ziele und Aufgaben einer Architektur festzulegen.

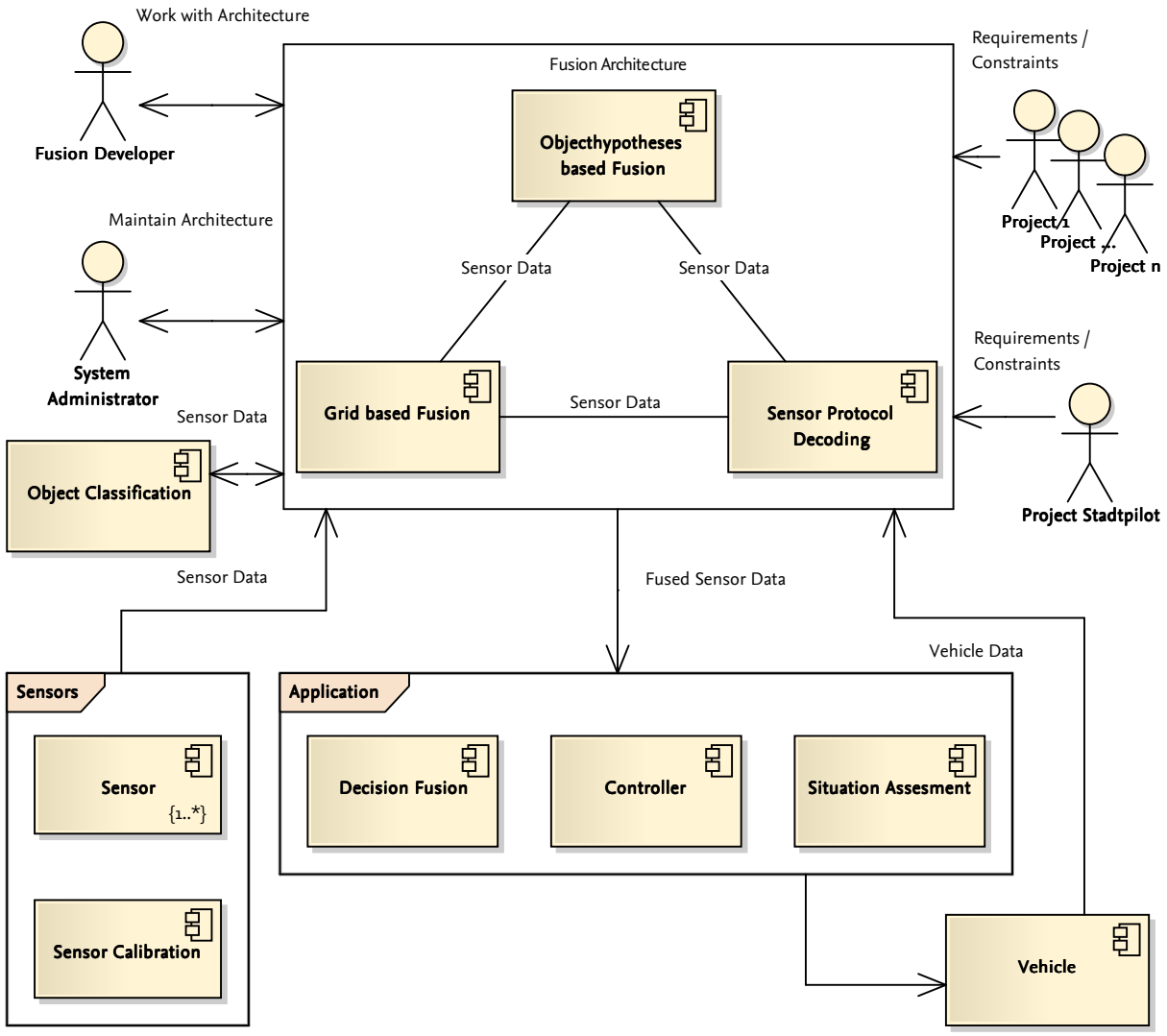


Abbildung 6.1: Kontextdiagramm der Architektur zur Sensordatenfusion

Die Architektur zur Fusion von Umfeld wahrnehmenden Sensoren interagiert mit einer Vielzahl von Stakeholdern (siehe Abbildung 6.1). Außerhalb der Systemgrenzen der Sensordatenfusionsarchitektur liegen zunächst die genutzten Sensoren selbst. Das System empfängt lediglich deren Messdaten über eine Schnittstelle, dekodiert die entsprechenden Protokolle, führt ggf. eine Vorverarbeitung durch und bearbeitet diese anschließend in einer Sensordatenfusion. Der Bereich der Sensorkalibrierung wird ebenfalls nicht innerhalb des Systems behandelt, da die dabei zu ermittelnden Größen lediglich Konfigurationsparameter aus Sicht der Fusionsarchitektur darstellen. Systeme zur Entscheidungsfusion werden auch nicht betrachtet. Sie gehören zu den Abnehmern (*Applikation*) der erzeugten Daten und interagieren über eine Schnittstelle mit den Modulen der Architektur.

Einige Architekturen zur Sensordatenfusion sehen den Bereich der Klassifikation von Objekthypothesen ebenfalls als eine Kernaufgabe einer Sensordatenfusion (z. B. Darms (2007, Seite 63)). Die Klassifikation von Objekthypothesen ist für eine Applikation oft eine wichtige Größe. Die Art und Weise und nach welchen Eigenschaften klassifiziert wird

bzw. welche Eigenschaften zu einer Klassifikation führen, sind jedoch hochgradig abhängig von der späteren Verwendung. Hier eine einheitliche Schnittstelle zu definieren und eine einheitliche Verarbeitung vorzuschlagen, führt entweder zu einem sehr abstrakten Element in der Architektur oder würde vielen Applikationen nicht gerecht werden. Im Rahmen der hier betrachteten Architektur wird die Klassifikation von Objekthypothesen deshalb außerhalb der Systemgrenzen gesehen.

Neben den technischen Systemen interagiert die Architektur während ihrer Realisierung mit den Entwicklern einer konkreten Fusion und den Systemadministratoren der Versuchsträger. Sie entwickeln einzelne Algorithmen, kombinieren vorhandene neu oder pflegen die Systemumgebung, um die Anforderungen einer Applikation zu erfüllen.

7 Anforderungen an die Softwarearchitektur

Auf der Basis des Architekturkontexts und der Zielformulierung können Anforderungen an die Softwarearchitektur zur Fusion von Umfeld wahrnehmenden Sensoren definiert werden. Diese stellen die Grundlage für die anschließende Entwicklung und spätere Realisierung der Architektur dar. Sie beschreiben die Wünsche und Bedürfnisse der Stakeholder genauso wie Anforderungen, die sich aus der Einsatzdomäne ableiten lassen.

> Eine Anforderung ist [dabei]:
>
> 1. Eine Bedingung oder Fähigkeit, die von einem Benutzer (Person oder System) zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird.
> 2. Eine Bedingung oder Fähigkeit, die ein System oder Teilsystem erfüllen oder besitzen muss, um einen Vertrag, eine Norm, eine Spezifikation oder andere, formell vorgegebene Dokumente zu erfüllen.
> 3. Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft gemäß (1) oder (2).
>
> (Übersetzung von Pohl u. Rupp (2011, Seite 11) nach NORM IEEE Std 610.12 (1990, Seite 62))

Anforderungen können in eine Reihe von Kategorien eingeteilt werden (Sommerville, 2004, Seite 117ff). Im Folgenden werden zunächst funktionale Anforderungen beschrieben. Sie beschreiben Funktionen, die das System für einen Stakeholder bereitstellt. Darauf folgend werden nicht-funktionale Anforderungen vorgestellt. Diese beschreiben extrafunktionale Eigenschaften der Architektur. Gefolgt werden sie von einem Abschnitt über Randbedingungen. Anforderungen, die sich nicht direkt an die Architektur, sondern an die spätere konkrete Umsetzung im Projekt Stadtpilot richten, werden in Abschnitt 13 beschrieben, da dieser Abschnitt nicht speziell auf ein Projekt zugeschnitten ist.

Die gefundenen Anforderungen können nicht nur nach funktional/nicht-funktional unterteilt werden, sondern auch nach den Quellen der Anforderungen. Systemanforderungen beschreiben dabei Eigenschaften, die durch die Entwickler des Systems in den Prozess eingebracht werden. Sie sind in der Entwicklersprache beschrieben und gehen explizit auf die Umsetzung einer Nutzeranforderung im System ein. Nutzeranforderungen resultieren aus den Wünschen von nicht direkt an der Entwicklung beteiligten Stakeholdern. Sie beschreiben meistens externe Abläufe oder Systemeigenschaften. Die dritte Gruppe von Anforderungen stellen die Domänenanforderungen und Randbedingungen dar. Sie beschreiben Forderungen, die sich aus der Anwendungsdomäne ergeben. Hier werden Forderungen zur Umsetzung von Standards, Normen oder grundlegenden Algorithmen erhoben.

Funktionale Anforderungen

FA1 Unterschiedliche Sensorkonfigurationen müssen genutzt werden können.

Die zu entwickelnde Architektur richtet sich nicht an eine spezielle Sensorkonfiguration, da, je nach Versuchsträger, unterschiedliche Konfigurationen zu erwarten sind. Die Architektur darf hier keine wesentlichen Einschränkungen vornehmen, die bestimmte Sensoren von vornherein ausschließen würden.

FA2 Die Fusionsstruktur muss allgemein übliche Abläufe aus der automotiven Umfeldwahrnehmung abbilden.

Das Domänenwissen über eine Sensordatenfusion ist über einen langen Zeitraum konstant und findet sich üblicherweise in jedem Fusionssystem wieder. Dieses muss durch die Architektur abgebildet werden, um den eigentlichen Zweck zu erfüllen.

FA3 Der Einsatz der Architektur in einem Projekt, das keine Sensordatenfusion benötigt, muss gewährleistet sein.

Der Kontext der Architektur enthält auch die Sensorprotokolldekodierung. Da nicht jedes Projekt eine Sensordatenfusion benötigt (z. B., wenn nur ein Sensor genutzt wird), muss auch die einfache Nutzung der Sensorprotokolldekodierung ohne aufwendige Sensordatenfusion möglich sein.

FA4 Es muss eine gitterbasierte Fusion unterstützt werden.

Eine häufig eingesetzte Form der Sensordatenfusion ist die gitterbasierte Fusion. Da die Architektur nicht für ein spezielles Projekt entwickelt wird, muss sich dieser Typ in der Architektur abbilden lassen. Darüber hinaus sollte auf die speziellen Anforderungen hinsichtlich Datenzugriff und -verwaltung Rücksicht genommen werden.

FA5 Es muss eine objekthypothesenbasierte Fusion unterstützt werden.

Eine häufig eingesetzte Form der Sensordatenfusion ist die objekthypothesenbasierte Fusion. Da die Architektur nicht für ein spezielles Projekt entwickelt wird, muss dieser Typ durch die Architektur abgebildet werden. Anforderungen hinsichtlich eines effizienten Datenzugriffs und einer effizienten Datenverarbeitung sollten ebenfalls berücksichtigt werden.

FA6 Die Sensorkonfiguration muss übermittelt werden.

Befestigungsort und Erfassungsbereiche eines Sensors oder einer Fusionsinstanz stellen eine wichtige Information für eine verarbeitende Applikation dar. Aus diesem Grund müssen diese Informationen übermittelt werden.

FA7 Das Alter der Objekthypothesen Daten muss übermittelt werden.

Für eine echtzeitfähige Applikation ist das Alter eines Datums von erheblicher Bedeutung. Veraltete Daten können zu Fehlentscheidungen oder bei Projektionen in die Zukunft von Objekthypothesen zu falschen Ergebnissen führen.

FA8 Der Erstellungszeitpunkt einer Objekthypothese muss übermittelt werden.

Zur Bestimmung der Qualität einer Objekthypothese stellt das Alter und damit die Stabilität der Hypothese einen erheblichen Anhaltspunkt dar.

FA9 Eine Objekthypothese muss eindeutig identifiziert werden.

Um eine Objekthypothese von anderen unterscheiden zu können, wird jeder Hypothese eine eindeutige Nummer zugewiesen.

FA10 Hierarchische Fusionsstrukturen müssen umsetzbar sein.

Sensordatenfusionen lassen sich auf unterschiedliche Weise konzipieren. Eine Variante ist, mehrere Fusionen in einer hierarchischen Struktur zu kombinieren (Darms, 2007, Seite 25ff).

FA11 Unterschiedliche Software- und Hardware-Schnittstellen (z. B. CAN, LIN, FlexRay, Ethernet) müssen transparent genutzt werden können.

Umfeld wahrnehmende Sensoren verfügen über eine Vielzahl von unterschiedlichen Hardware-Schnittstellen, Software-Bibliotheken und Protokollen. Die Architektur muss mit üblichen Bussystemen umgehen können und gleiche Protokolle über unterschiedliche Hardware-Schnittstellen empfangen können.

FA12 Die Daten der Sensoren sowie die Ergebnisse der Umfeldwahrnehmung müssen aufgezeichnet werden können.

Zur späteren Auswertung und zur Dokumentation von Testfahrten ist eine Aufzeichnung der Daten der Sensoren genauso wie der Ergebnisse der Umfeldwahrnehmung notwendig.

Nicht-funktionale Anforderungen

NF13 Die Unabhängigkeit der einzelnen Basisfunktionen und Algorithmen innerhalb der Sensordatenfusion soll zum Zwecke der Austauschbarkeit gewährleistet sein.

Die in Abschnitt 8 beschriebenen Basisfunktionen sollen über einheitliche Schnittstellen entkoppelt werden. Je mehr der implementierten Algorithmen in einer späteren Realisierung der Architektur wiederverwendet werden können, desto größer ist der Nutzen für den Entwickler. Aus diesem Grund soll die Architektur den Entwickler bei der Wiederverwendung von Algorithmen unterstützen. Ziel ist es, Basisfunktionen in anderen Projekten wieder verwenden zu können.

NF14 Der Einsatz in unterschiedlichen Projekten soll mit geringem Aufwand möglich sein.

Versuchsträger werden oft in mehreren Projekten eingesetzt. Da unterschiedliche Projekte teilweise verschiedene Anforderungen und Schnittstellen haben, ist es wünschenswert, einen möglichst geringen Anpassungsaufwand beim Einsatz der Sensordatenfusion in einem anderen Projekt zu haben. Dies kann durch die Anpassung der Schnittstellen, Änderung von Konfigurationsparametern, aber auch durch die Zusammenstellung einer neuen

Umfeldwahrnehmung aus den bereits implementierten Einzelkomponenten geschehen.

NF15 Das Objekthypothesenmodell und das Gitterzellenmodell muss flexibel sein.

Als allgemeine Architektur zur Sensordatenfusion darf sich das System nicht auf eine spezielle Beschreibung des Bewegungs-, Geometrie- und Gitterzellenmodells festlegen. Je nach Anforderungen der Applikation und den Fähigkeiten der Sensoren können die Modelle sehr unterschiedliche Größen aufweisen. Außerdem soll die Möglichkeit bestehen, zusätzlich beliebige Daten einer Objekthypothese zuzuordnen.

NF16 Die eingesetzten Fusionsalgorithmen werden nicht durch die Architektur festgelegt.

Zur Entwicklung von Sensordatenfusionen werden eine Vielzahl von unterschiedlichen Algorithmen eingesetzt. Je nach den Fähigkeiten der Sensoren und den Anforderungen der Applikation sind andere Algorithmen einzusetzen. Die Architektur darf hier möglichst keine Voraussetzungen schaffen, die der Verwendung üblicher Algorithmen entgegenstehen.

NF17 Die Fusionsarchitektur darf von sich aus keine größeren zusätzlichen Latenzen erzeugen.

In einem Echtzeitsystem ist das Alter von Daten von großer Bedeutung. Aus diesem Grund sollen Latenzen, bedingt durch die Architektur, möglichst vermieden werden. Latenzen, bedingt durch den Einsatz von spezifischen Verarbeitungsalgorithmen, fallen hier nicht unter diese Anforderung.

NF18 Der Zugriff der Applikation auf Gitterdaten muss effizient sein.

Gitterdaten benötigen bereits bei kleinen repräsentierten Flächen und groben Auflösungen einen großen Speicherplatz. Damit eine Applikation diese Daten nutzen kann, muss die Übertragungsform effizient gestaltet oder der Zugriff auf den originalen Speicherplatz sichergestellt werden.

Randbedingungen

RB19 Das Anwendungsframework ist ADTF.

Im Rahmen des Projekts Stadtpilot und damit auf dem Versuchsträger wird das Automotive Data and Time-triggered Framework (ADTF) eingesetzt (Audi Electronic Venture GmbH, 2011). Die zu entwickelnde Architektur und die spätere Realisierung müssen sich in dieses Framework einfügen.

RB20 Das Kommunikationsframework ist RTI DDS.

Zur Kommunikation zwischen Applikationsgrenzen im Versuchsträger des Stadtpilot-Projekts wird das Data Distribution System (DDS) von Real-Time Innovations Inc. eingesetzt (Real-Time Innovations Inc., 2011). Zur Sicherstellung der Kompatibilität mit anderen Applikationen, ist es zu verwenden.

RB21 Die Entwicklungssprache ist C++.

Bedingt durch die Festlegung auf das ADTF ist die Entwicklungssprache C++. Der Sprachumfang kann voll genutzt werden.

RB22 Die Architektur ist an die Automotive-Domäne gebunden.

Der Begriff Sensordatenfusion wird in vielen unterschiedlichen Bereichen eingesetzt. Die hier beschriebene Architektur richtet sich ausschließlich an den Bereich der Sensordatenfusion von Umfeld wahrnehmenden Sensoren der Automotive-Domäne.

Anforderung	Quelle:			Anforderung	Quelle:		
	System	Nutzer	Domäne		System	Nutzer	Domäne
FA1		✓		FA12		✓	
FA2			✓	NF13		✓	
FA3		✓		NF14		✓	
FA4			✓	NF15		✓	
FA5			✓	NF16		✓	
FA6		✓		NF17	✓	✓	
FA7		✓		NF18		✓	
FA8		✓		RB19	✓		
FA9		✓		RB20	✓		
FA10		✓		RB21	✓		
FA11			✓	RB22			✓

Tabelle 7.1: Quellen der aufgestellten Architektur Anforderungen

8 Logische Architektur der Fusionsstruktur

Die Basis einer jeden Architektur bildet das Wissen über die Domäne, deren Abläufe und Strukturen, in der es angewendet werden soll (Buschmann u. a. (2007, Seite 182ff) oder Starke u. Hruschka (2011, Seite 43)). Dieses Wissen ist oft über einen sehr langen Zeitraum konstant und zeichnet sich durch Technologie- genauso wie Projektunabhängigkeit aus. Basierend auf den in den Abschnitten 2.2, 2.3 und 2.5 vorgestellten Algorithmen und Architekturen wird im Folgenden eine logische Architektur einer Sensordatenfusion aus einzelnen Basisfunktionen vorgestellt (Anforderung FA2).

Allgemein kann die Struktur einer Sensordatenfusion auf vier Basisfunktionen zurückgeführt werden:

- Datenspeicherung
- Selektion
- Abstraktion
- Fusion

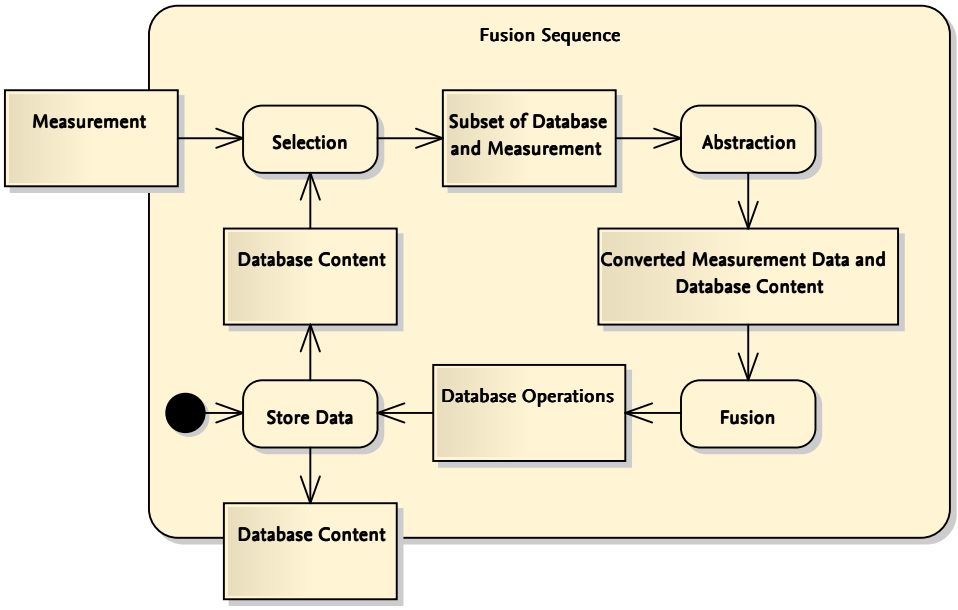


Abbildung 8.1: Aktivitätsdiagramm eines Fusionszyklus mithilfe von Basisfunktionen. Datentypen zwischen den Aktivitäten werden über Objektknoten dargestellt.

Eine Sensordatenfusion beschreibt dabei immer einen zyklischen Ablauf, welcher eingehende Messdaten verarbeitet. Als Messdaten wird jedes Datum S angesehen, das von außerhalb in den Fusionszyklus eingebracht wird. Innerhalb des Zyklus werden sogenannte Tracks verarbeitet. Ein Track T stellt dabei die kleinste zu verarbeitende Einheit dar und wird

beschrieben durch einen Zustandsvektor. In Abbildung 8.1 wird der Fusionszyklus, bestehend aus den Basisfunktionen, im Zusammenhang gezeigt. Je nach Sensordatenfusion können diese Basisfunktionen in unterschiedlicher Ausprägung oder auch mehrfach innerhalb des Zyklus' vorhanden sein. Ein Beispiel hierzu findet sich am Ende dieses Kapitels.

Die *Datenspeicherung* verwaltet alle Tracks, die Teil des Fusionszyklus sind, und stellt Lese- und Schreibfunktionen für die anderen Basisfunktionen zur Verfügung. Hierzu wird eine Kopie des Datenbestandes an die Aktivität *Selektion* gesand.

Die Basisfunktion *Selektion* findet sich am Anfang des Fusionszyklus. Die Eingabe dieser Funktion bildet neben den Messdaten die Menge der Tracks der *Datenspeicherung*. Das Ziel ist die Auswahl einer Menge von Tracks und deren Zuordnung zu den Messdaten. Die Ausgabe der Funktion ist eine Menge von Tracks, die einem Messdatum oder mehreren Messdaten mit einer gewissen Qualität zugeordnet wurden. Die Menge der Tracks kann hierbei auch die Leere-Menge sein. Übliche Algorithmen zur *Selektion* sind der Nearest-Neighbour-Algorithmus (Blackman u. Popoli, 1999, Seite 9f), der Munkres-Algorithmus (Munkres, 1957) oder der Bresenham-Algorithmus (Bresenham, 1965). Zusätzlich kommen während der Durchführung der Algorithmen oft Metriken zur Anwendung, auf deren Grundlage die Zuordnung durchgeführt wird.

Diese Untermenge des Datenbestandes, die Messdaten und die zugehörigen Qualitätswerte werden von der *Abstraktion* verarbeitet. Die *Abstraktion* bildet ein Messdatum auf die Eingangsgrößen der *Fusionsfunktion* ab. Die Ausgabe ist die Kombination aus Track bzw. der Leeren-Menge, Messdatum, Selektionsqualität und Eingangsgröße der Kombinationsfunktion. Üblicherweise beschreibt diese Funktion die Messmatrix eines Filterprozesses oder, im Fall von der gitterbasierten Sensordatenfusion, die Messfunktion.

Die durch die *Abstraktion* erzeugten Daten werden nun miteinander kombiniert. Das Ergebnis stellt einen veränderten Track sowie eine Operation in der *Datenspeicherung* dar. Der Begriff „veränderter Track“ schließt erneut die Leere-Menge mit ein. Auf diese Weise lässt sich neben der Aktualisierung auch die Erzeugung, bzw. im Fall der Leeren-Menge als Messdatum, auch die Entfernung eines Tracks abbilden. Übliche Algorithmen zur *Fusion* stellen beispielsweise das Kalman-Filter (Kalman, 1960), das Partikel-Filter (Gordon u. a., 1993) oder das Dempster-Shafer-Filter (Shafer, 1976) dar.

8.1 Anwendung auf objekthypothesen- und gitterbasierte Fusionen

Aus dieser logischen Architektur lassen sich konkrete Sensordatenfusionen ableiten. Im Folgenden wird je eine Zusammenstellung der Basisfunktionen für die Anwendung als objekthypothesen- und der gitterbasierten Sensordatenfusion vorgestellt. Beide Fusionstypen werden im automotiven Kontext aktiv in einer breiten Variantenvielfalt eingesetzt (siehe Abschnitt 2.1 und Abschnitt 2.5).

Objekthypothesenbasierte Fusion

In einer objekthypothesenbasierten Datenfusion werden Messdaten durch Objekthypothesen repräsentiert. Jede Objekthypothese repräsentiert die Annahme über die Existenz eines realen Objekts, welches durch einen Zustandsvektor beschrieben wird.

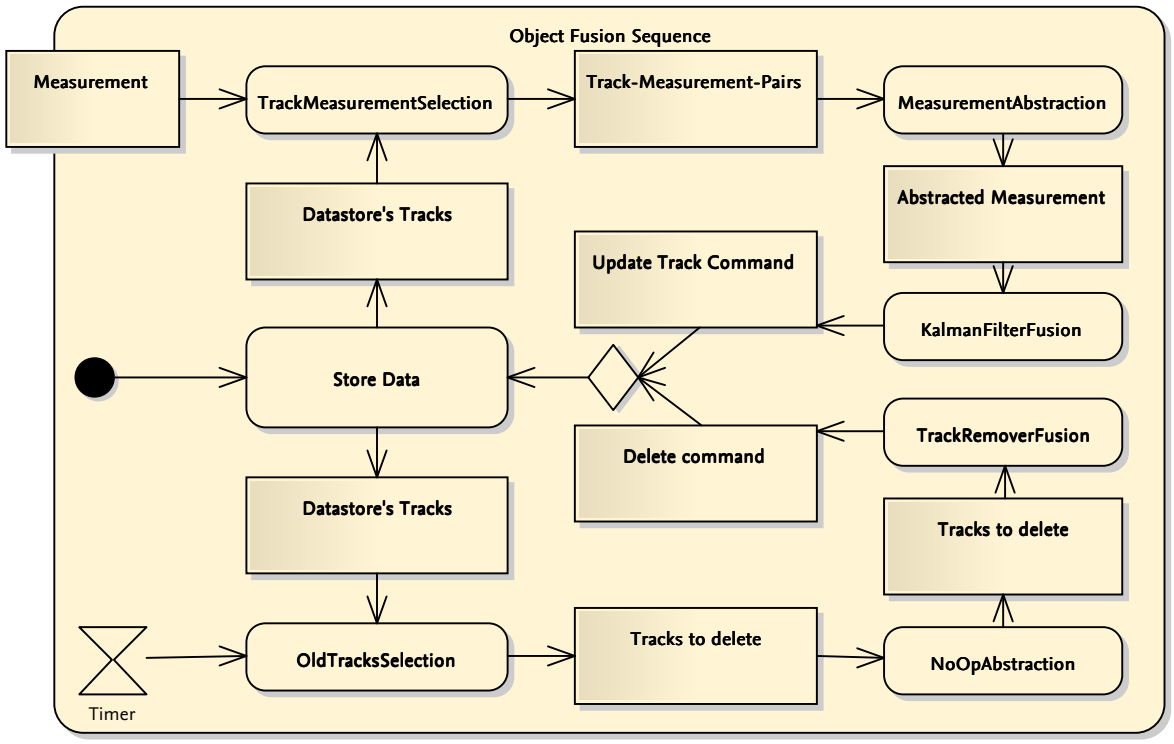


Abbildung 8.2: Beispiel: Aktivitätsdiagramm einer objekthypothesenbasierten Sensordatenfusion aus Basisfunktionen

Um zu zeigen, wie eine objekthypothesenbasierte Datenfusion mithilfe der Basisfunktionen zu realisieren ist, wird diese im Folgenden anhand eines akademischen Beispiels beschrieben. Für dieses wird angenommen, dass die Datenfusion die Messdaten eines Sensors verarbeitet, der Objekthypothesen in Polarkoordinaten beschreibt (z. B. ein Lidarsensor). Eine Applikation, die auf der Fusion aufbaut, benötigt jedoch einen Zustandsvektor in kartesischen Koordinaten und darüber hinaus eine Geschwindigkeitsinformation des Tracks in X- und Y-Richtung. Die Umformung des Zustandsvektors und die Berechnung der Geschwindigkeitsinformation werden durch die Sensordatenfusion vorgenommen. Der Zustandsvektor des Tracks besteht aus einer Position im kartesischen Raum und der zugehörigen Geschwindigkeitsinformation.

Erzeugt der Sensor nun ein Messdatum, so wird dieses zunächst über einen *Selektionsalgorithmus* mit den in der *Datenspeicherung* abgelegten Tracks verglichen und entsprechend zugeordnet (siehe Abbildung 8.2). Das Track-Messdatum-Paar kann nun der *Abstraktionsfunktion* übergeben werden. Diese formt den Zustandsvektor in die Eingangsgrößen des Filters um. Anschließend liegt die Position des Messdatums in kartesischen Koordinaten vor und kann, z. B. durch ein Kalman-Filter, verarbeitet werden. Nach der *Filterfunktion* wird der aktualisierte Track zusammen mit dem Update-Kommando der *Datenspeicherung* übergeben.

Bewegen sich Objekte aus dem Erfassungsbereich des Sensors heraus, können sie nicht länger wahrgenommen werden. Der Track, der dieses Objekt repräsentiert, wird jedoch zunächst in der *Datenspeicherung* vorgehalten. Um diese nicht länger notwendigen Daten zu entfernen, lässt sich ein weiterer Fusionszyklus der Datenfusion hinzufügen. Als Messdatum

dient diesmal ein durch einen Taktgeber erzeugtes Ereignis. Dieses wird durch eine *Selektionsfunktion* verarbeitet, die aus der *Datenspeicherung* alle Tracks auswählt, welche über einen gewissen Zeitraum nicht aktualisiert wurden. Die *Abstraktionsfunktion* reduziert die Daten der ausgewählten Tracks auf ein eindeutiges Identifikationsmerkmal. Im *Fusionsschritt* wird für die betreffenden Tracks ein Löschkommando erzeugt, welches letztlich durch die *Datenspeicherung* umgesetzt wird.

Gitterbasierte Fusion

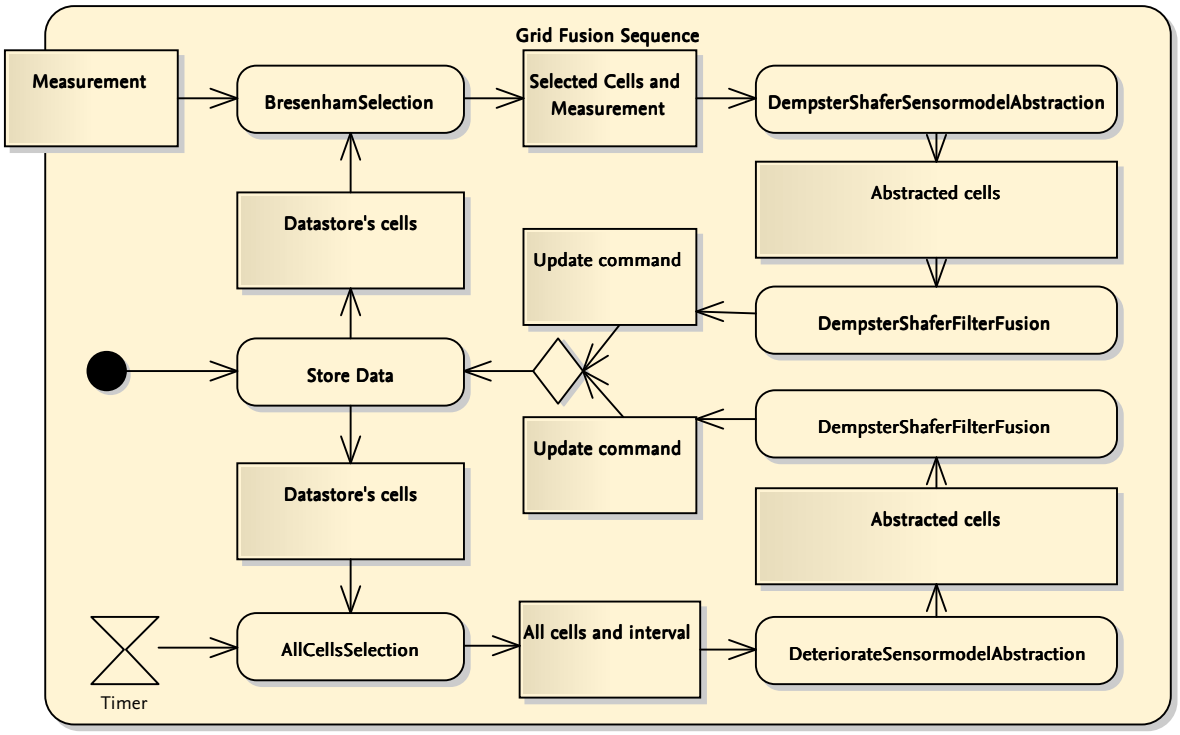


Abbildung 8.3: Beispiel: Aktivitätsdiagramm einer gitterbasierten Sensordatenfusion aus Basisfunktionen

Im Vergleich zur objekthypothesenbasierten Fusion beschreibt die gitterbasierte Fusion eine bestimmte Fläche der Fahrzeugumgebung, bestehend aus Gitterzellen. Diese Zellen enthalten je einen Zustandsvektor und repräsentieren im Fall der gitterbasierten Fusion die Tracks. Wie im vorangegangenen Abschnitt wird auch für die gitterbasierte Datenfusion die Realisierung mithilfe der Basisfunktionen anhand eines akademischen Beispiels beschrieben (siehe Abbildung 8.3). In diesem Beispiel wird die Anzahl der Gitterzellen als konstant angenommen. Daraus resultiert eine Vorinitialisierung der *Datenspeicherung* mit der entsprechenden Anzahl an Tracks. Verarbeitet werden sollen die Messdaten eines Lasersensors, welcher die Entfernung zu einem Reflektionspunkt misst. Da ein Laserstrahl normalerweise keine Hindernisse durchdringt, kann angenommen werden, dass der Pfad vom Sensorursprung zum Reflektionspunkt nicht durch Objekte blockiert wird. Liegt nun ein Messdatum des Sensors vor, so wählt die *Selektionsfunktion* mithilfe des Bresenham-

Algorithmus (Bresenham, 1965) alle Tracks (Zellen) auf dem Pfad vom Sensorursprung zum Reflektionspunkt aus und verknüpft sie mit dem Messdatum. Anschließend werden sie durch eine *Abstraktionsfunktion* auf die Eingangsgrößen des Filters abgebildet. Für dieses Beispiel wurde ein Dempster-Shafer-Filter gewählt, welches einen Wahrscheinlichkeitspotenzraum bestehend aus den Elementen „belegt“, „frei“ und „unbekannt“ besitzt. Für jeden Track wird auf Basis des Messdatums ein entsprechendes 3-Tupel bestimmt. Dieses kann anschließend in der *Filterfunktion* mit dem Zustand des Tracks fusioniert und in der *Datenspeicherung* aktualisiert werden.

Analog zur objekthypothesenbasierten Fusion lässt sich auch der gitterbasierten Fusion ein zweiter Fusionszyklus hinzufügen. Nimmt der Sensor eine Reflektion nicht länger wahr, so werden die Tracks, die zuvor noch aktualisiert wurden, u. U. nicht länger in die Verarbeitung einbezogen. In diesem Fall sollte die Evidenz für die Elemente „belegt“ und „frei“ gegen null gehen, während die Evidenz für „unbekannt“ steigen sollte. Dies lässt sich beispielsweise zeitgesteuert umsetzen. Erneut gibt ein Taktgeber ein Ereignis vor. Der *Selektionsalgorithmus* wählt für dieses spezielle Beispiel alle Tracks aus der *Datenspeicherung* aus und übergibt sie an die *Abstraktionsfunktion*. Als Messwert liegt nur das zeitliche Ereignis vor. Aus diesem Grund kann für alle Tracks die gleiche Massenverteilung im Wahrscheinlichkeitspotenzraum an die *Filterfunktion* übergeben werden. Diese setzt genau die gleiche Funktion wie im Falle eines vorhandenen Messwerts um und erzeugt aktualisierte Tracks für die *Datenspeicherung*.

9 Softwarearchitektur - Bausteinsicht

Zur Beschreibung der konkreten Softwarearchitektur wird zunächst auf die Bausteinsicht eingegangen. Diese bildet nach Starke u. Hruschka das „Kernstück der Softwarearchitektur“ (Starke u. Hruschka, 2011, Seite 30). In ihr werden die statischen Elemente der Architektur sowie ihre Beziehungen untereinander beschrieben. Elemente der Bausteinsicht können z. B. Klassen, Pakete oder andere Artefakte sein. Diese Bestandteile kommunizieren über angebotene und benötigte Schnittstellen miteinander. Ihr innerer Aufbau kann dabei zur besseren Abstraktion zunächst verborgen bleiben (Blackbox) und in einer späteren Präzisierung detaillierter beschrieben werden.

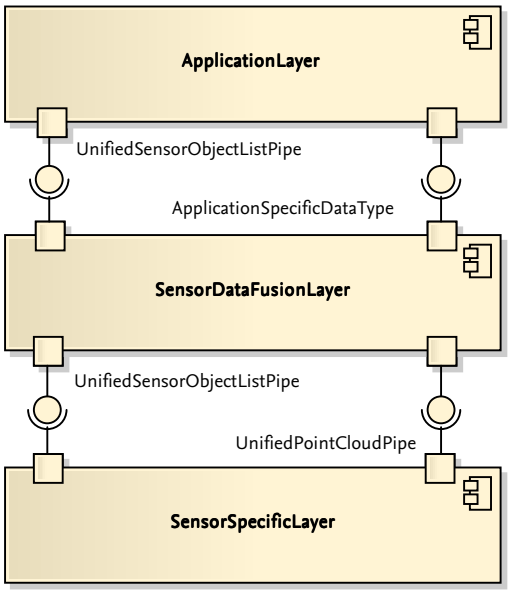


Abbildung 9.1: Übersicht über die Ebenenstruktur der Architektur

Die in den vorangegangenen Abschnitten entwickelten Anforderungen sowie das aus Abschnitt 8 bekannte Domänenwissen werden im Folgenden in eine konkrete Softwarearchitektur überführt. Zur Erhöhung der Übersichtlichkeit und zur Entkopplung von Elementen (Anforderungen FA1, NF14) wird die Softwarearchitektur in drei Pakete aufgeteilt (siehe Abbildung 9.1).

Die drei Pakete *Sensor Specific Layer*, *Sensor Data Fusion Layer*, *Application Layer* bilden das Layer-Architekturmuster ab und widmen sich unterschiedlichen Aufgaben. Der *Sensor Specific Layer* stellt die Anbindung von Sensoren über einheitliche Schnittstellen zur Verfügung (Anforderung FA1), während der *Sensor Data Fusion Layer* sich mit der Umsetzung des Domänenwissens aus Abschnitt 8 befasst (Anforderung FA2). Der *Application Layer* beschreibt die konsumierende Applikation und damit die Datensenke für die erzeugten Daten (Anforderung NF14). Wie in Abbildung 9.1 zu sehen ist, kann der *Sensor Specific Layer* nicht ohne Einbezug des *Sensor Data Fusion Layers* direkt mit dem *Application*

Layer kommunizieren. Dies widerspricht zunächst Anforderung FA3. Die Umsetzung der Anforderung lässt sich jedoch mithilfe eines Fusionsmoduls realisieren, das lediglich die eingehenden Daten weiterleitet. Zwischen den einzelnen Paketen finden vor allem zwei Datenstrukturen Verwendung: *UnifiedPointCloud* und *UnifiedSensorObjectList*. Darüber hinaus sind zwischen *Sensor Data Fusion Layer* und *Application Layer* auch spezielle Datenstrukturen zur besseren Übertragung von Ergebnissen einer Auswertung von Gitterdaten vorgesehen (Anforderung NF18).

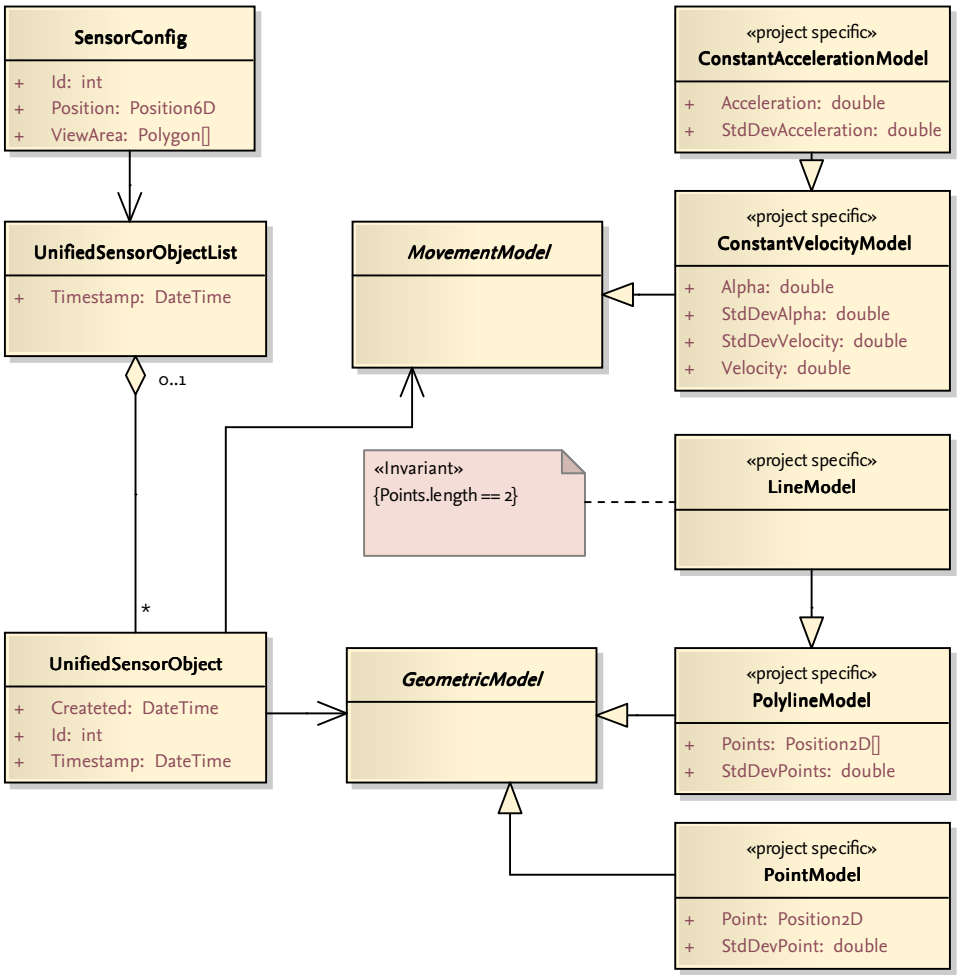


Abbildung 9.2: Datenstruktur *UnifiedSensorObjectList*

Die Datenstruktur *UnifiedSensorObjectList* ist in Abbildung 9.2 dargestellt. Sie enthält neben der *SensorConfig*, welche Informationen über den Befestigungsort und die Erfassungsbereiche (Anforderung FA6) enthält, eine Liste von Objekthypothesen (*UnifiedSensorObjects*). Jedes *UnifiedSensorObject* besteht aus einer eindeutigen Nummer (Anforderung FA9), dem Alter der Daten (Anforderung FA7), dem Erstellungszeitpunkt der Hypothese (Anforderung FA8) sowie einem Geometrie- und einem Bewegungsmodell (Anforderung NF15). Jede dieser Klassen (*MovementModel*, *GeometricModel*) ist abstrakt definiert und wird anschließend durch Ableitungen konkretisiert. In Abbildung 9.2 sind einige Ableitungen beispielhaft dargestellt. Da sie projektbezogen umgesetzt werden, sind sie mit dem

Stereotype ≪project specific≫ gekennzeichnet. Sie bilden drei Geometrie- und zwei Bewegungsmodelle ab. Je nach eingesetzten Sensoren und Anforderungen der Applikation können die Klassen beliebig erweitert werden. Durch die Aufspaltung von Geometrie- und Bewegungsmodell können verarbeitende Algorithmen einfacher wiederverwendet werden, da, wenn sie sich z. B. nur auf das Geometriemodell beziehen, auch mit Objekthypothesen mit anderen Bewegungsmodellen genutzt werden können (Anforderung NF14).

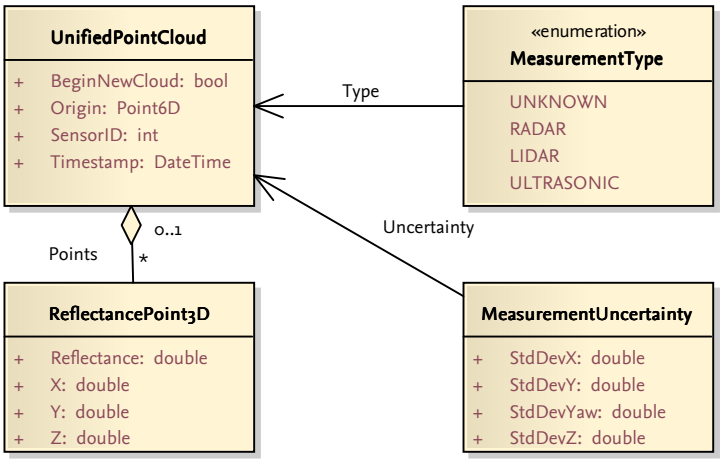


Abbildung 9.3: Datenstruktur *UnifiedPointCloud*

Die zweite Datenstruktur *UnifiedPointCloud* (siehe Abbildung 9.3) beschreibt Messpunktwolken. Dabei enthält die Struktur einen Messzeitpunkt (Anforderung FA7), einen Befestigungsort (Anforderung FA6), die Identifikationsnummer des Sensors, die Messunsicherheit über alle Messpunkte und eine Liste der Messpunkte. Darüber hinaus wird das Messverfahren der Daten mit gespeichert, da es später erheblichen Einfluss auf die Verarbeitung innerhalb des *Sensor Data Fusion Layers* haben kann. Ein einzelner Messpunkt enthält eine 3D-Koordinate in einem kartesischen Koordinatensystem sowie einen Messwert für die Reflektivität des Messpunkts. Da Punktwolken möglicherweise eine sehr große Zahl an Punkten enthalten können, ist die Aufteilung einer Messpunktwolke in mehrere einzelne Punktwolken möglich. Hierzu kann der Beginn einer Reihe von Messpunktwolken markiert werden.

9.1 Sensor Specific Layer

In einem Umfeldwahrnehmungssystem können Daten unterschiedlicher Sensoren verarbeitet werden (Anforderung FA1). Jeder Sensor hat meist eine andere Schnittstellendefinition und wird i. d. R. durch ein eigenes Protokoll angesprochen. Damit die höheren Schichten die Daten verarbeiten können, wird der Datenstrom eines Sensors, je nach Sensorfähigkeiten, in eine der beiden Datenstrukturen *UnifiedPointCloud* oder *UnifiedSensorObjectList* konvertiert. Darüber hinaus müssen Sensoren oft mit Daten über die Eigenbewegung des Versuchsträgers versorgt oder andere zyklische Aufgaben übernommen werden.

Die Klassen innerhalb des *Sensor Specific Layers* übernehmen die Dekodierung von Sensor spezifischen Protokollen sowie die Stimulation von Sensoren (z. B. mit Fahrzeugeigendaten). Zur Umsetzung von Anforderung FA11 wird zunächst eine Abstraktion von den Hardwareschnittstellen durchgeführt. Hierzu werden die empfangenen Datenpakete in die Struktur

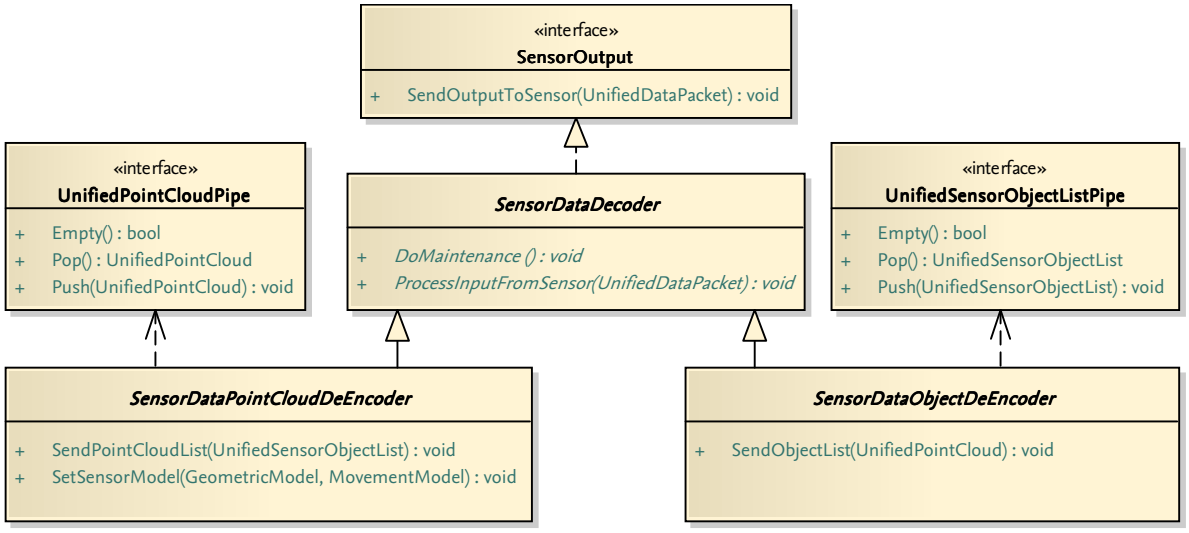


Abbildung 9.4: Klassendiagramm zur Sensoranbindung

UnifiedDataPacket überführt. Die Datenstruktur *UnifiedDataPacket* enthält dabei den Empfangszeitpunkt des Pakets, den Typ des Datenstroms sowie ein Binärpaket mit den empfangenen Daten. Die weitere Verarbeitung nutzt diese Datenstrukturen und kann so ein Protokoll, das über unterschiedliche Hardwareschnittstellen übertragen wird (z. B. beim IBEO Alaska XT), auf die gleiche Weise verarbeiten.

Des weiteren sind alle Datenstrukturen, die zwischen den Paketen genutzt werden, durch die Interface Definition Language des Data Distribution Service (Real-Time Innovations Inc., 2011) beschrieben. Durch diese systemunabhängige Beschreibung können die Daten innerhalb eines DDS-Netzwerks auch anderen Systemen verfügbar gemacht oder Komponenten ausgetauscht werden. So ist es beispielsweise möglich, die Daten zu visualisieren oder sie zur Archivierung abzuspeichern (Anforderung FA12).

Zur eigentlichen Dekodierung von Sensorprotokollen steht die abstrakte Basisklasse *SensorDeEncoder* bereit. Die zentrale Funktion ist dabei *ProcessInputFromSensor*. Sie erhält den Inhalt eines *UnifiedDataPacket*, damit dieser in einer sensorspezifischen Implementierung dekodiert werden kann. Der Empfang von *UnifiedDataPackets* sowie eine Reihe von Servicefunktionen sind ebenfalls verfügbar. So stehen Funktionen zur Transformation von Koordinatensystemen und grundlegende Geometriealgorithmen bereit. Neben der Dekodierung kann der *SensorDeEncoder* auch eine zyklisch aufgerufene Funktion bereitstellen, um eine Stimulation des Sensors oder Verwaltungsaufgaben durchzuführen.

Die beiden Klassen *SensorDataObjectDeEncoder* und *SensorDataPointCloudDeEncoder* werden von der Klasse *SensorDeEncoder* abgeleitet. Sie bieten je eine Schnittstelle an, um auf die dekodierten Sensordaten in Form von *UnifiedSensorObjectLists* oder *UnifiedPointClouds* zuzugreifen. Darüber hinaus wird im Fall des *SensorDataObjectDeEncoders* durch die Funktion *SetSensorModel* auch das Geometrie- und Bewegungsmodell des Sensors festgelegt. Die Funktionen *SendObjectList* bzw. *SendPointCloudList* setzen allgemeine protokollunabhängige Elemente in den Datenstrukturen (z. B. die Identifikationsnummer des Sensors) und stellen die Daten anschließend an den externen Schnittstellen zur Verfügung.

Abbildung 9.5 beschreibt die Ableitungsstruktur eines IBEO Alaska XT-Sensors und eines

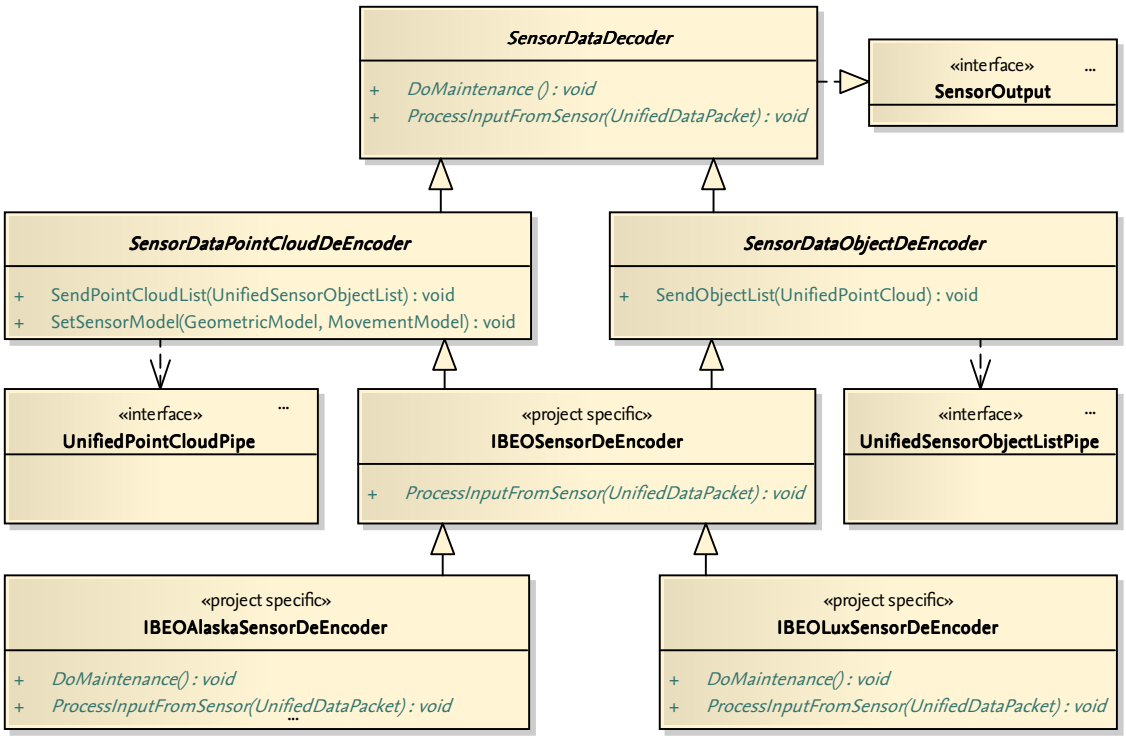


Abbildung 9.5: Darstellung der Anbindung von Sensoren am Beispiel der Sensoren IBEO Alaska XT und IBEO Lux

IBEO Lux-Sensors. Zu sehen sind hier die Basisklassen *SensorDeEncoder* mit seinen Kindsklassen *SensorDataObjectDeEncoder* und *SensorDataPointCloudDeEncoder*. Von diesen beiden ist die Klasse *IBEOSensorDeEncoder* abgeleitet, da diese speziellen Sensoren sowohl eine Objekthypothesenbildung durchführen als auch einzelne Messpunkte bereitstellen. In der Klasse *IBEOSensorDeEncoder* wird die Paketdekodierung des Sensorprotokolls durchgeführt, um die ähnliche Paketdekodierung der beiden Sensoren zentral zu implementieren. Anschließend wird die Klasse weiter zu den Klassen *IBEOAlaskaXTSensorDeEncoder* und *IBEOLuxSensorDeEncoder* spezialisiert, welche die Unterschiede in der Dekodierung des Protokolls umsetzen und die Stimulation mit Fahrzeugeigendaten übernehmen.

9.2 Sensor Data Fusion Layer

Der *Sensor Data Fusion Layer* widmet sich der Umsetzung der logischen Architektur einer Fusionsstruktur aus Abschnitt 8 (Anforderung FA2). Nach den Anforderungen FA4 und FA5 sollen zwei verschiedene, in der automotiven Domäne übliche, Typen von Sensordatenfusionen umgesetzt werden. Die objekthypothesen- und die gitterbasierte Sensordatenfusion lassen sich zwar beide auf die logische Architektur einer Fusionsstruktur abbilden, da sie jedoch ein erheblich unterschiedliches Laufzeitverhalten aufweisen, werden sie durch die Architektur getrennt dargestellt.

Je nach Anwendungsfall ist es notwendig, unterschiedliche Fusionsimplementierungen einzusetzen. Im Sinne der Architektur werden sie alle innerhalb des *Sensor Data Fusion Layers* angesiedelt. Jede konkrete Implementierung stellt ein für sich abgeschlossenes Fusionsmodul dar. Die Fusionsmodule kommunizieren über die bereits beschriebenen Schnittstellen (siehe Abbildungen 9.2 und 9.3). Da die Schnittstellen für Ein- und Ausgabe identisch angelegt sind, ist eine Kombination von Fusionsmodulen möglich.

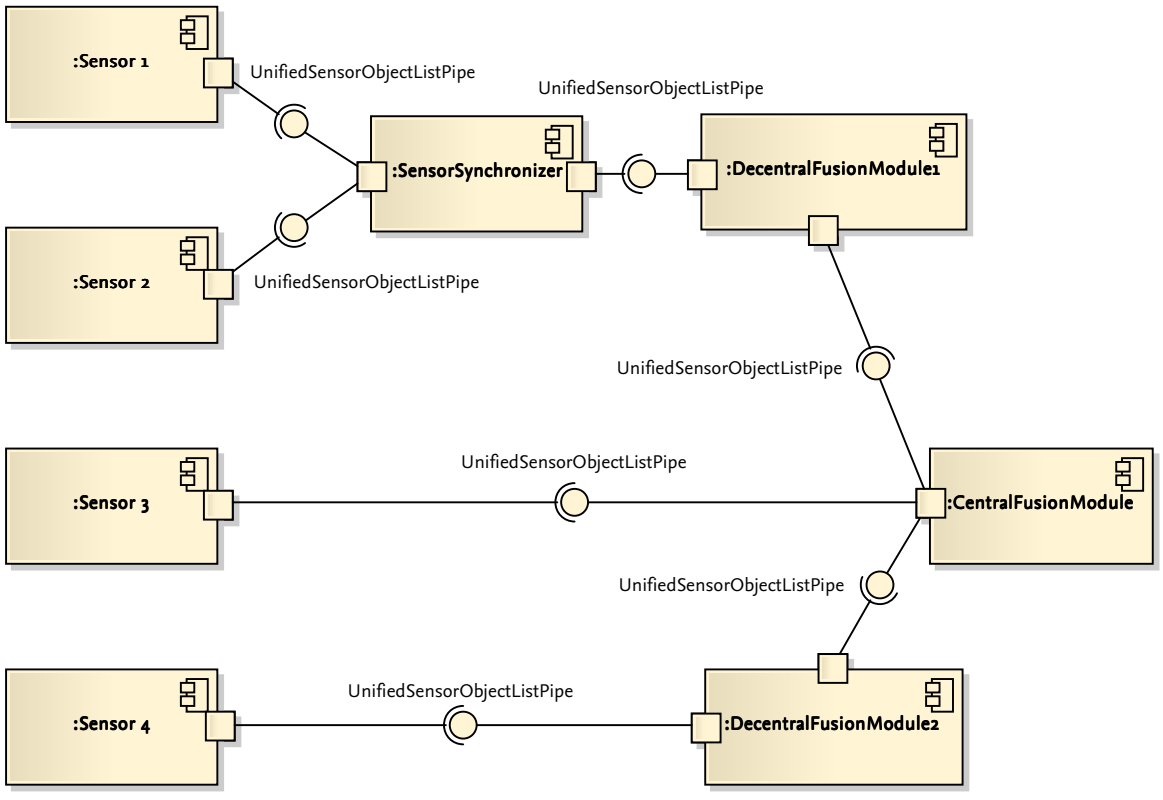


Abbildung 9.6: Beispiel zur Kombination von unterschiedlichen Fusionsmodulen

Werden mehrere voneinander abhängige Fusionsmodule eingesetzt, so bilden sich zwangsläufig Hierarchien heraus. Nach Darms kann die Zusammenstellung aller genutzten Fusionsmodule dabei nach den Kriterien zentral/dezentral und synchron/asynchron unterschieden werden (Darms, 2007, Seite 25ff). Durch die Kombination von Fusionsmodulen lassen sich diese Typen mit der Architektur abbilden (Anforderung FA10). Abbildung 9.6 zeigt, wie mithilfe eines Synchronisationsmoduls zunächst ein synchron arbeitendes Fusionsmodul

betrieben wird. Dieses liefert zusammen mit einem weiteren die Eingangsdaten für ein zentral arbeitendes Fusionsmodul.

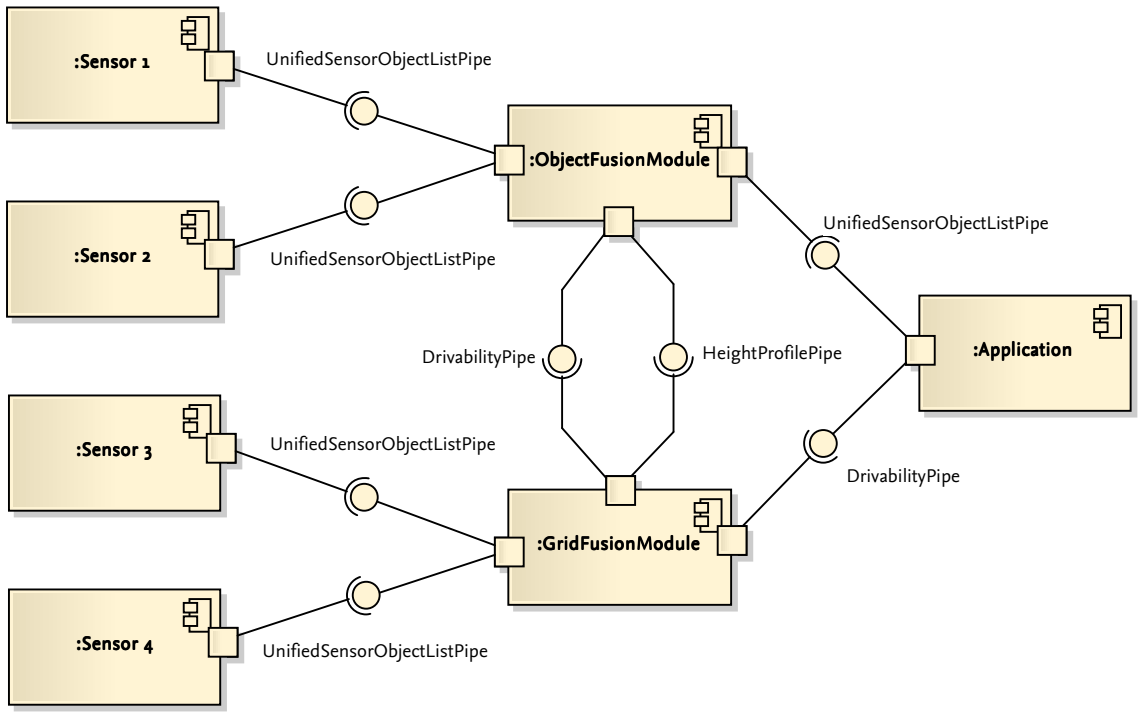


Abbildung 9.7: Darstellung der Kombination von gitter- und objekthypothesenbasierter Fusion nach Effertz (2008)

Vorstellbar ist beispielsweise die Kombination von gitter- und objekthypothesenbasierten Fusionsmodulen, wie sie z. B. von Effertz (2008) vorgeschlagen wird (siehe Abbildung 9.7). Dabei werden einige Sensoren dazu genutzt, um eine Befahrbarkeitsanalyse sowie eine Höhenkarte der Fahrzeugumgebung aufzubauen. Diese Informationen werden anschließend zur Selektion von für die Applikation relevanten Objekthypothesen genutzt. Sie reduzieren so den Rechenaufwand innerhalb der objekthypothesenbasierten Fusion.

9.2.1 Objekthypothesenbasierte Fusion

Ein objekthypothesenbasiertes Fusionsmodul teilt sich in fünf einzelne Komponenten auf und implementiert die in Abschnitt 8 vorgestellte Fusionsstruktur. Der Architektur der Komponenten liegt der Architekturstil der Event-Driven-Architecture nach Schatten u. a. (2010, Seite 221) zugrunde. Dabei wird innerhalb eines Systems ein Zustand definiert, der sich durch externe Ereignisse ändert. Als Ereignis wird in diesem Fall der Empfang einer Objekthypothesenliste oder der Ablauf einer Zeitspanne angesehen. Beide Ereignisse führen zur Ausführung von Algorithmen welche i. d. R. den Zustand des Fusionsmoduls ändern. Zur Verarbeitung eines Events kommt eine Pipes-and-Filter-Struktur zum Einsatz und ermöglicht so eine Abgrenzung der einzelnen Komponenten voneinander.

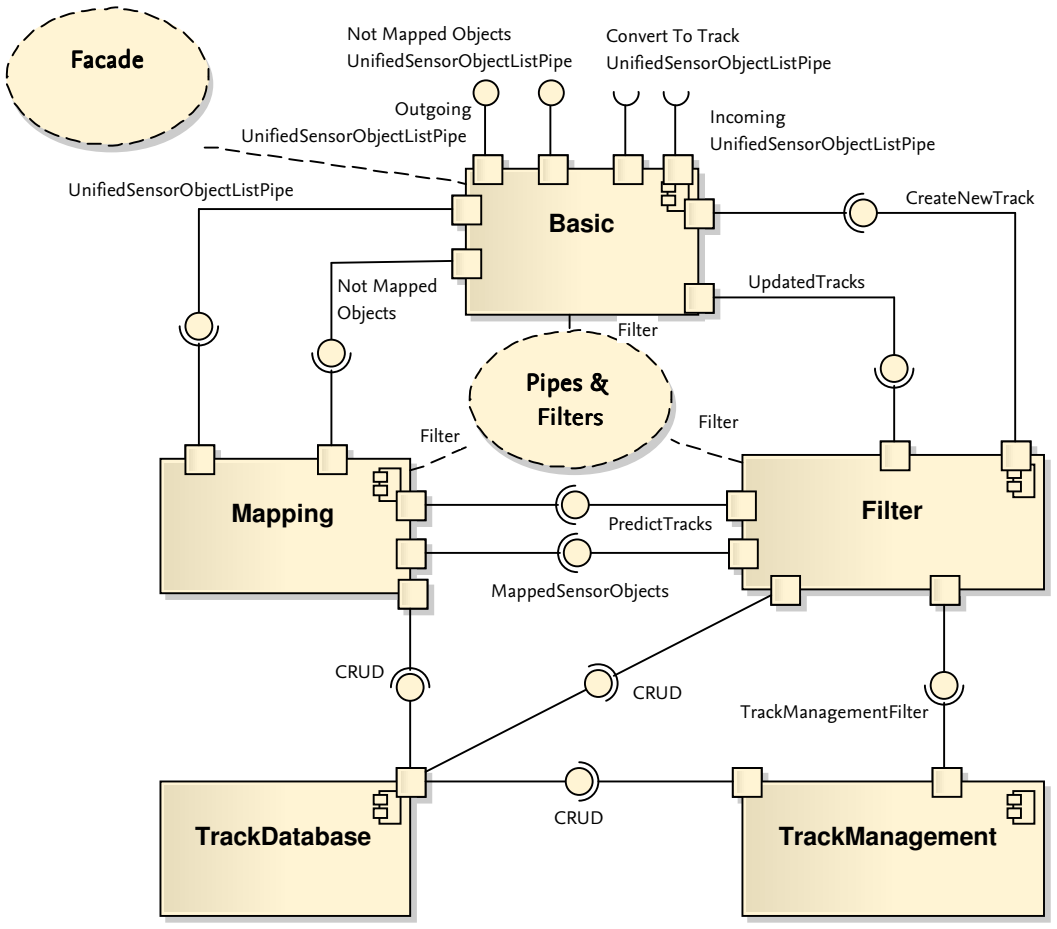


Abbildung 9.8: Komponentendiagramm eines Fusionsmoduls

Die Komponenten *Basic*, *Mapping*, *Filter*, *TrackDatabase* und *TrackManagement* werden in Abbildung 9.8 gezeigt. Zu sehen sind zudem die jeweiligen Schnittstellen zwischen den Komponenten sowie die verwendeten Entwurfsmuster. Im Folgenden wird auf die Komponenten näher eingegangen.

Komponente: *Basic*

Die *Basic*-Komponente ist nicht Teil der logischen Architektur einer Fusionsstruktur und stellt Servicefunktionen für die anderen Komponenten zur Verfügung. Sie implementiert das Entwurfsmuster Fassade. Die Schnittstellen für Empfang und Versand von Objekthypothesenlisten werden durch diese Komponente bereitgestellt. Darüber hinaus ist die Ablaufsteuerung des Fusionsprozesses in der Verantwortung der *Basic*-Komponente. Auch die Kommunikation mit dem ADTF (Anforderung RB19) und den anderen Komponenten ist nur über diese Komponente möglich. So können dem ADTF beispielsweise Statistikdaten über den Fusionsprozess zur Verfügung gestellt oder Ereignisse des Frameworks an die internen Komponenten weitergeleitet werden.

Neben den Schnittstellen für Empfang und Versand von Objekthypothesenlisten wird ein weiterer Datenstrom bereitgestellt. Dieser enthält alle nicht-bearbeitbaren Objekthypothesen. Im Falle der direkten Rückkopplung an die Eingangsschnittstelle der *Basic*-Komponente führen die gesendeten Objekthypothesen direkt zu neuen Tracks. Wird der Datenstrom jedoch gefiltert, so lässt sich das Erzeugen von neuen Tracks verhindern oder verzögern. Ein Algorithmus, der diese Möglichkeiten nutzt, wird in Abschnitt 14.3 vorgestellt. Darüber hinaus ist auch die Vorinitialisierung der *TrackDatabase* mit Tracks darüber möglich.

Komponente: *Mapping*

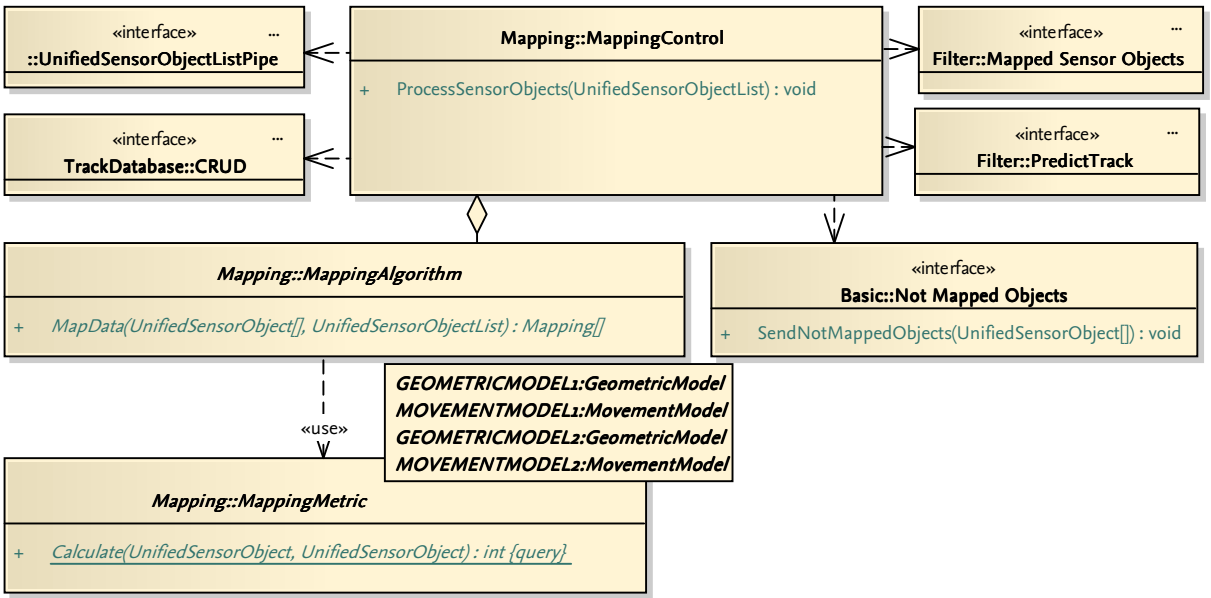


Abbildung 9.9: Klassen innerhalb der Komponente *Mapping*

Die *Mapping*-Komponente teilt sich in drei Klassen auf. Die Klasse *Control* stellt die Schnittstelle zu den anderen Komponenten zur Verfügung. Hier werden vor allem Mengenoperationen auf den Objekthypothesenlisten durchgeführt. Die eingehenden Objekthypothesen werden mithilfe der Klassen *MappingAlgorithm* und *MappingMetric* in zwei Gruppen aufgeteilt: in Objekthypothesen, die bekannten Tracks zugeordnet werden konnten, und in

solche Objekthypothesen nicht zugeordnet werden können. Zugeordnete Objekthypothesen werden zusammen mit dem zugeordneten Track und einem Maß für die Qualität der Zuordnung an die Komponente *Filter* übergeben. Nicht zuordbare Objekthypothesen werden als *UnifiedSensorObjectList* der *Basic*-Komponente zum Versand überlassen.

Durch die Aufteilung der Komponente in drei einzelne Klassen ist ein großes Maß an Wiederverwendung möglich (Anforderung NF13). So lassen sich sowohl Algorithmen als auch Metriken miteinander kombinieren und an die Anforderungen anpassen, ohne z. B. die Schnittstellen und den grundlegenden Ablauf neu entwickeln zu müssen.

Die Klasse *MappingAlgorithm* implementiert einen Zuordnungsalgorithmus (z. B. Auktionsalgorithmus (Bertsekas, 1991) oder ungarischer Algorithmus (Munkres, 1957)). Die Funktion *MapData* erhält eine Liste aller gemessenen Objekthypothesen sowie aller Tracks. Diese werden durch den implementierten Algorithmus unter Verwendung einer Metrik einander zugeordnet. Das Ergebnis sind 3-Tupel, bestehend aus Objekthypothese, Track und Zuordnungsqualität.

Um eine Zuordnung zwischen Messdatum und Track zu bestimmen, sind eine oder mehrere Metriken notwendig (z. B. euklidischer Abstand (Deza u. Deza, 2009, Seite 104) oder Mahalanobis Abstand (Mahalanobis, 1936)). Je nach verwendetem Geometrie- und Bewegungsmodell sowie den Anforderungen an das Umfeldwahrnehmungssystem können sie sich erheblich unterscheiden. Die Klasse *MappingMetric* abstrahiert diese Diversität zwischen den Metriken durch C++-Templates. Auf diese Weise ist es notwendig, für jede Kombination von Geometrie- und Bewegungsmodellen eine spezielle Implementierung der Metrik vorzunehmen oder entsprechende Ableitungsstrukturen bereitzustellen und so eine Generalisierung der Metriken zu erreichen.

Komponente: *Filter*

Die *Filter*-Komponente (siehe Abbildung 9.10) stellt eine Schlüsselkomponente innerhalb der Verarbeitung der Messdaten dar. Sie definiert nicht nur das Geometrie- und Bewegungsmodell der Tracks des Fusionsmoduls, sondern stellt auch eine ganze Reihe von Funktionen den anderen Komponenten zur Verfügung (z. B. Prädiktion einer Objekthypothese auf einen anderen Zeitpunkt).

Drei Anwendungsfälle richten sich an die *Filter*-Komponente: die Erstellung von neuen Objekthypothesen, die Prädiktion einer Objekthypothese auf einen anderen Zeitpunkt und die Fusion von Messdaten mit einem Track. Dazu ist die Komponente in drei Klassen aufgeteilt: Die *FilterControl*-Klasse übernimmt die Kommunikation mit den anderen Komponenten, die Klasse *FilterAlgorithm* definiert das Filter und die Klasse *FilterUpdate* beschreibt die Konvertierung eines Messdatums in den Zustandsvektor eines neuen Tracks sowie die Abstraktion des Messwerts.

Die *FilterControl*-Klasse der *Filter*-Komponente empfängt die Kombination aus Track und gemessener Objekthypothese und übergibt diese an die *FilterAlgorithm*-Klasse zur Fusion. Anschließend werden die Ergebnisse in der *TrackDatabase*-Komponente gespeichert sowie über eine Schnittstelle der *Basis*-Komponente zur Verfügung gestellt.

Innerhalb der *FilterAlgorithm*-Klasse wird das Filter zur Fusion von Messdaten mit Tracks implementiert. Hierzu können drei Schnittstellen genutzt werden: *PredictTrack*, *CreateNewTrack* und *TrackManagementFilter*. Die *CreateNewTrack*-Funktion der *Create-*

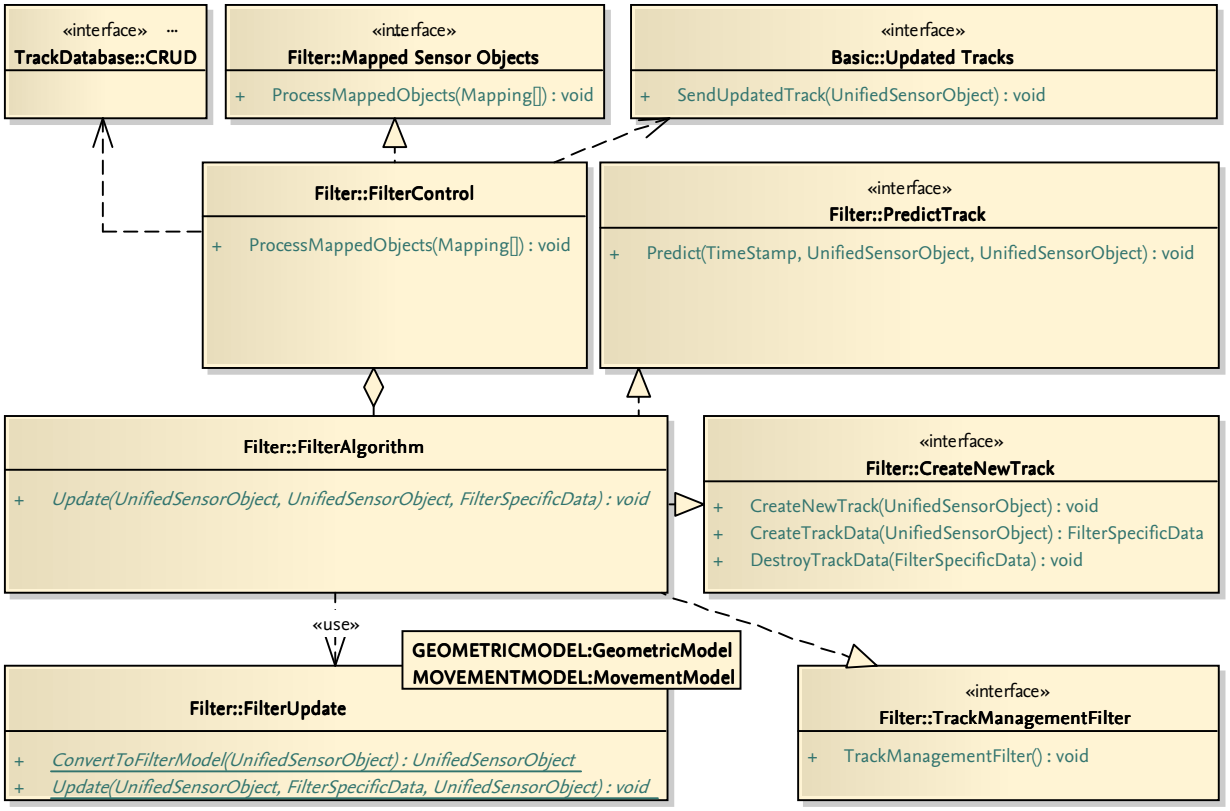


Abbildung 9.10: Klassen innerhalb der Komponente *Filter*

NewTrack-Schnittstelle erzeugt einen neuen Track aus einer gemessenen Objekthypothese. Hierzu wird zunächst das Geometrie- und Bewegungsmodell mithilfe einer Instanz der Klasse *FilterUpdate* in das zum Filter gehörige Modell konvertiert. Anschließend wird zusätzlich eine interne filterspezifische Datenstruktur *TrackData* über die *CreateTrackData*-Funktion angelegt. In ihr können beliebige Daten gespeichert werden, die nicht Teil der Datenstruktur *UnifiedSensorObject* sind (z. B. Kreuzkovarianzen oder Heuristiken). Die *Predict*-Funktion der *PredictTrack*-Schnittstelle wird von der *Mapping*- und *Basis*-Komponente genutzt und bildet einen Track auf einen neuen Zeitpunkt ab. Dabei wird eine Kopie des Tracks mit den neuen Daten angelegt, der dann z. B. bei der Objekthypothesenzuordnung genutzt wird. Zur Fusion von Messdaten mit Tracks wird die Funktion *Update* durch die *FilterControl*-Klasse genutzt. Sie nutzt die Klasse *FilterUpdate* zur Abstraktion der Messdaten auf die Eingangsgrößen des Filters (z. B. Belegung der Messmatrix des Kalman-Filters). Diese Daten werden anschließend genutzt, um den eigentlichen Fusionsalgorithmus durchzuführen. Neben der Erzeugung von neuen Tracks, der Prädiktion und der Fusion wird innerhalb der *FilterAlgorithm*-Klasse auch die Schnittstelle *TrackManagementFilter* implementiert. Diese stellt filterspezifische Teile der Komponente *TrackManagement* bereit. Hier könnte z. B. ein Algorithmus ablaufen, der Tracks nach ungültigen Zuständen absucht und diese korrigiert.

Wie bereits erwähnt, bildet die Klasse *FilterUpdate* einerseits gemessene Objekthypothesen auf Tracks ab und abstrahiert andererseits Messgrößen auf die Eingänge des Filters. Die Klasse besitzt zwei Template-Parameter zur Unterscheidung der Geometrie- und Bewegungsmodelle. Auf diese Weise können Template-Instanzen spezialisiert oder allgemeiner genutzt

und so der Wiederverwendungsgrad erhöht werden (Anforderung NF13). Die Funktion *ConvertToFilterModel* bildet eine Objekthypothese von dem Geometrie- und Bewegungsmodell der Template-Parameter auf die Modelle der zugehörigen *FilterAlgorithm*-Klasse ab. Zur Fusion von Messdaten mit einem Track wird die Funktion *Update* genutzt. Sie bildet zunächst die Messgrößen der Objekthypothese auf die Eingangsgrößen des Filters ab (z. B. mithilfe einer Messmatrix im Falle eines Kalman-Filters). Anschließend wird der Fusionsschritt entsprechend des Filters ausgeführt und die Ergebnisse im Track gespeichert.

Weitere Komponenten

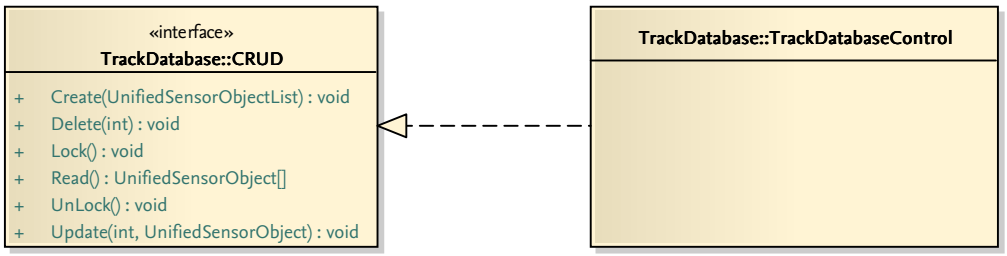


Abbildung 9.11: Klasse innerhalb der Komponente *TrackDatabase*

Neben den bereits vorgestellten Komponenten existieren die Komponenten *TrackDatabase* und *TrackManagement* innerhalb eines objekthypothesenbasierten Fusionsmoduls. Die einem Fusionsmodul bekannten Tracks werden in der Komponente *TrackDatabase* gespeichert. Damit realisiert die Komponente den Teil *Datenspeicherung* der logischen Architektur der Fusionsstruktur aus Abschnitt 8. Alle anderen Komponenten können über die Schnittstelle CRUD (Create, Read, Update, Delete) auf die Daten zugreifen. Neben der Speicherung der Daten wird auch der Zugriff auf die Daten über Methoden der Prozesssynchronisation geregelt. Die Klasse *TrackDatabaseControl* implementiert innerhalb der Komponente die Schnittstellen und realisiert die Speicherung der Daten. Gespeichert wird dabei neben den Tracks auch die filterspezifische Datenstruktur eines Tracks, eine Identifikationsnummer (Anforderung FA9), der Zeitpunkt der Erstellung (Anforderung FA8) sowie der Zeitpunkt der letzten Aktualisierung (Anforderung FA7). Da die Komponente die authoritative Speicherung innerhalb eines Fusionsmoduls darstellt, übernimmt sie auch die Vergabe von Identifikationsnummern bei der Erstellung von neuen Tracks.

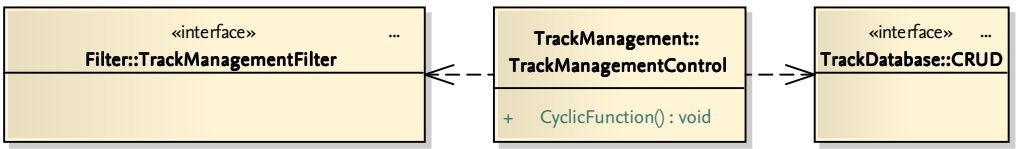


Abbildung 9.12: Klasse innerhalb der Komponente *TrackManagement*

Parallel zur messdatengetriebenen Verarbeitungskette innerhalb des Fusionsmoduls wird eine zweite zeitgesteuerte Kette vorgesehen. Die Komponente ist in Abbildung 9.12 dargestellt

und ist für die Pflege der Tracks zuständig. Hierbei wird zyklisch der Bestand an Tracks untersucht und ggf. Anpassungen vorgenommen. Unabhängig von der im Fusionsmodul eingesetzten *Filter*-Komponente lässt sich so z. B. das Entfernen von Tracks die längere Zeit nicht aktualisiert wurden, realisieren. Neben filterunabhängigen Algorithmen nutzt die *TrackManagement*-Komponente die Schnittstelle *TrackManagementFilter* der Komponente *Filter* zur filterspezifischen Pflege der Tracks.

9.2.2 Gitterbasierte Fusion

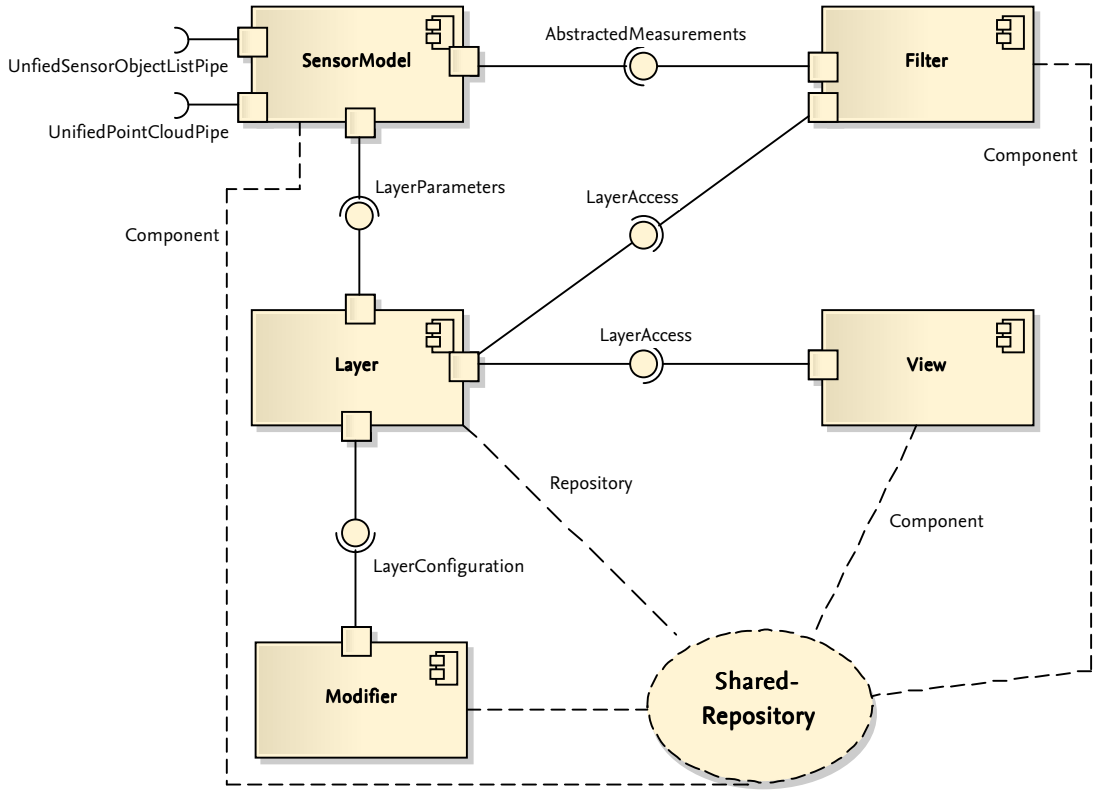


Abbildung 9.13: Komponentendiagramm eines gitterbasierten Fusionsmoduls

Wie die objekthypothesenbasierten Fusionsmodule teilen sich auch die gitterbasierten Fusionsmodule in mehrere Komponenten auf (siehe Abbildung 9.13) und implementieren die logische Architektur einer Fusionsstruktur aus Abschnitt 8. Das Design folgt dabei dem Shared-Repository-Entwurfsmuster. Dabei wird innerhalb der Komponente *Layer* ein zentraler Speicher eingerichtet, auf den die anderen Komponenten zugreifen können und ihre Arbeitsergebnisse abspeichern können.

Innerhalb eines gitterbasierten Fusionsmodules können alle Komponenten mehrfach vorkommen, um sich den Anforderungen der Applikation anzupassen oder unterschiedliche Eingangsdaten zu verarbeiten. So können beispielsweise in einem Modul drei Instanzen der *Layer*-Komponente vorkommen, in denen die Daten unterschiedlicher Sensortechnologien fusioniert werden (siehe Abbildung 9.14). Anschließend werden die Ergebnisse in einem Gesamt-Layer zusammengefasst und der Applikation präsentiert.

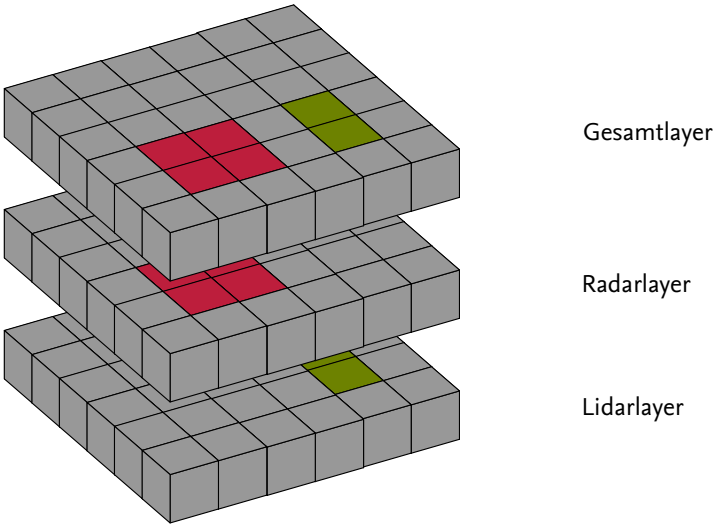


Abbildung 9.14: Logische Ebenen (*Layer*), z.B. Radarlayer, Lidarlayer und Gesamtlayer

Komponente: *Layer*

Die *Layer*-Komponente (siehe Abbildung 9.15) bildet die Objektspeicherung der logischen Architektur einer Fusionsstruktur ab und stellt zugleich das Shared-Repository-Entwurfsmuster dar. Wie bereits beschrieben, wird bei der gitterbasierten Datenfusion eine zwei-dimensionale Fläche der realen Welt durch äquidistante Zellen abgebildet. Dabei beschreibt das Adjektiv „äquidistant“ nicht notwendigerweise nur den euklidischen Abstand, sondern lässt sich auch auf radiale oder rechteckige Anordnungen anwenden. Jede dieser Zellen ist beschrieben durch einen Zustandsvektor (z. B. Belegungswahrscheinlichkeit oder Geschwindigkeitsvektor). Innerhalb einer *Layer*-Komponente werden die Zellen zum Zwecke des effizienteren Zugriffs in größeren Einheiten zu *GridBlocks* gruppiert. Ein *Layer* besteht damit aus einer ganzzahligen Menge von *GridBlocks*, die wiederum je aus der gleichen Anzahl von Zellen bestehen.

Neben der Speicherung der Gitterdatenstruktur übernimmt die Komponente auch die Zugriffskontrolle auf die Daten. Die auf den Layer zugreifenden Komponenten können gleichzeitig auf unterschiedlichen Teilen der Datenstruktur arbeiten. Die Auflösung ist hierbei durch die Größe der *GridBlocks* gegeben. Jeder *GridBlock* kann durch eine Komponente für den schreibenden bzw. lesenden Zugriff angefordert werden. Dabei sind auch mehrere lesende Zugriffe gleichzeitig möglich.

Zum Zugriff auf den *Layer* werden drei Schnittstellen angeboten: *LayerParameters*, *LayerAccess* und *LayerConfiguration*. Mithilfe der *LayerParameters*-Schnittstelle bekommt eine Komponente Zugriff auf die grundlegenden Eigenschaften eines *Layers* (z. B. Ausmaße oder Auflösung der Zellen). Die *LayerAccess*-Schnittstelle erweitert die *LayerParameters*-Schnittstelle, um die Möglichkeit einen spezifischen *GridBlock* anzufordern. Diese *GridBlock*-Instanz kann anschließend genutzt werden, um Daten auszulesen bzw. zu verändern. Abschließend steht die Schnittstelle *LayerConfiguration* zur Verfügung. Mit ihr ist es möglich, den *Layer* zu verschieben oder in seiner Größe zu verändern.

Da alle anderen Komponenten die *Layer*-Komponente nutzen, um ihre Daten zu speichern

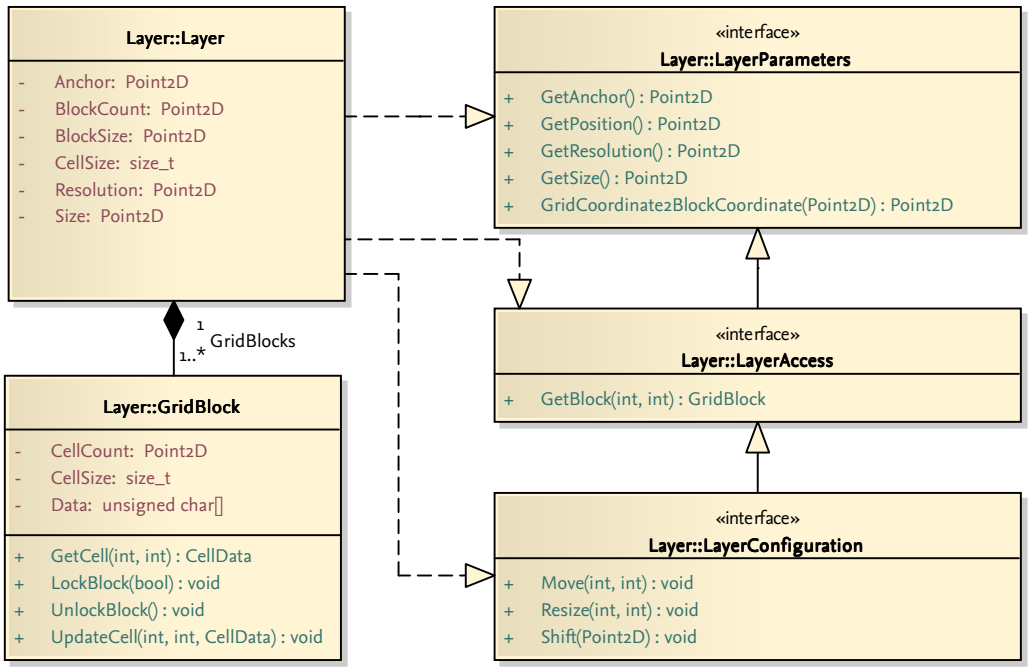


Abbildung 9.15: Klassendiagramm der Komponente *Layer*

bzw. Inhalte zu verändern, werden auch die für die Verarbeitung notwendigen Koordinatensysteme in dieser Komponente definiert. Die Position eines Layers ist relativ zu einem Anker definiert. Dieser Anker kann z. B. eine ortsfeste Koordinate (Weltkoordinaten) darstellen oder auch in einem beweglichen Koordinatensystem (Fahrzeugkoordinaten) definiert werden. Die Koordinate einer Zelle sowie die Verschiebung des *Layers* gegenüber dem Anker wird in Gitterkoordinaten angegeben. Durch den Einsatz eines festen Ankers bleiben die Koordinaten einer Zelle auch bei Umformung oder Translation eines *Layers* konstant. Zur Umrechnung der Koordinatensysteme durch die anderen Komponenten werden von der *LayerParameters*-Schnittstelle Funktionen zum Zugriff auf die Parameter bereitgestellt.

Komponente: *SensorModel*

Die *SensorModel*-Komponente bildet die Selektion sowie die Abstraktion der logischen Architektur einer Fusionsstruktur ab. Verarbeitet werden, je nach Sensorfähigkeiten, die Datenstrukturen *UnifiedPointCloud* oder *UnifiedSensorObjectList*. Innerhalb eines Algorithmus werden zunächst Zellen anhand eines Messdatums ausgewählt und anschließend mit einem sensortechnologiespezifischen Modell abstrahiert (z. B. Foessel (2000) oder Thrun u. a. (2005, Seite 288)). Gemein bleibt allen Algorithmen, dass die Messwerte auf eine Menge von zu aktualisierenden Zellen des *Layers* abgebildet werden. Ergänzt wird diese Menge durch einen für eine spezielle Instanz einer *Filter*-Komponente notwendigen Wert pro Zelle. Abbildung 9.16 stellt diese Datenstrukturen beispielhaft für die Filter-Typen Dempster-Shafer und Bayes dar. Die Liste der abstrahierten und somit zu aktualisierenden Zellen wird über die Schnittstelle *AbstractedMeasurements* der *Filter*-Komponente zur Verfügung gestellt.

Bestimmte Instanzen der Komponenten *SensorModel* und *Filter* stellen eine Art Tandem

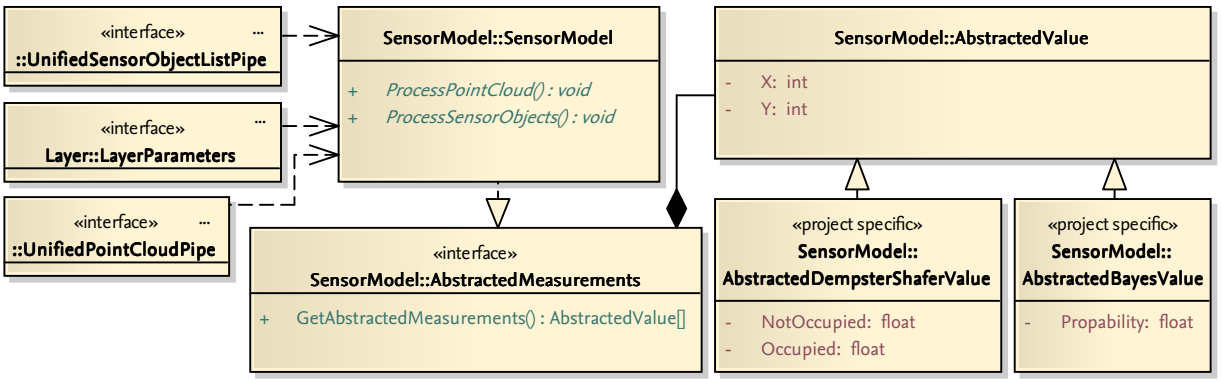


Abbildung 9.16: Klassendiagramm der Komponente *SensorModel*

dar. Durch die Abhängigkeit des Filters von dem Typ der abstrahierten Messdaten kann nicht jede *Filter*-Instanz mit jeder *SensorModel*-Instanz zusammen genutzt werden. So kann z. B. ein *SensorModel*, das seine Messwerte auf Werte für die Wahrscheinlichkeit der Existenz eines Hindernisses abbildet, nicht in Zusammenarbeit mit einem Dempster-Shafer-Filter genutzt werden, da Messvektoren eines Dempster-Shafer-Filters mindestens aus zwei Werten bestehen, deren Größen sich nicht unmittelbar umrechnen lassen.

Komponente: *Filter*

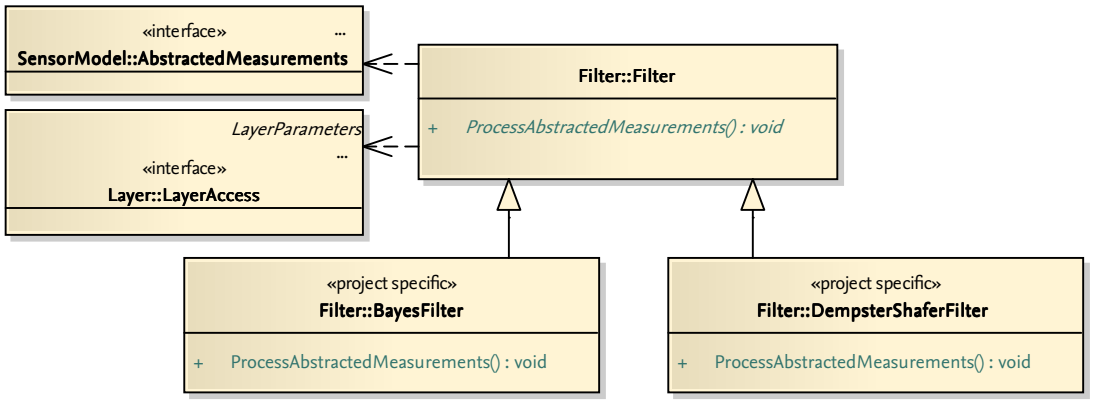


Abbildung 9.17: Klassendiagramm der Komponente *Filter*

Die Fusion der abstrahierten Messwerte der *Sensormodel*-Komponente geschieht durch eine Instanz der Komponente *Filter*. Innerhalb der Komponente wird der konkrete Filteralgorithmus implementiert. Dies kann beispielsweise ein binäres Bayes- oder ein Dempster-Shafer-Filter (Ribo u. Pinz, 2001) sein (siehe Abbildung 9.17).

Zur Kommunikation mit der *Layer*-Komponente wird die *LayerAccess*-Schnittstelle genutzt. Hierüber wird eine konkrete Zelle, für die ein abstrahierter Messwert vorliegt, ausgelesen. Ihr Zustand wird mit dem Messwert durch den implementierten Algorithmus kombiniert und das Ergebnis anschließend wieder in den *Layer* zurück geschrieben.

Komponente: *Modifier*

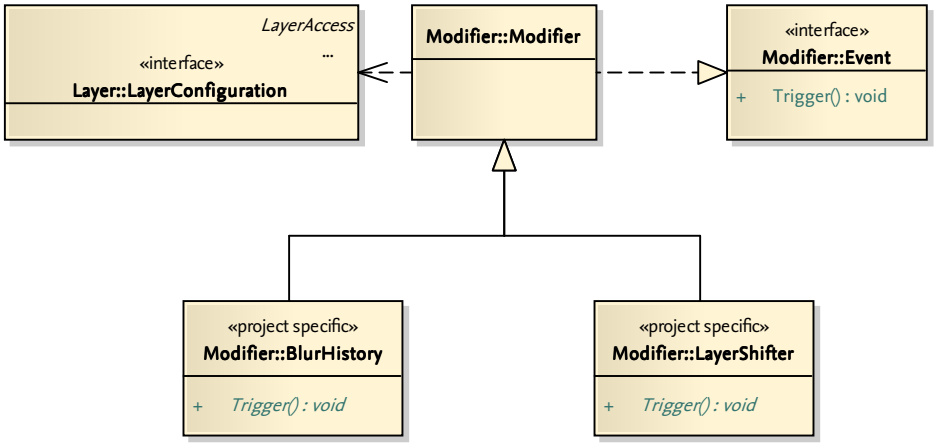


Abbildung 9.18: Klassendiagramm der Komponente *Modifier*

Auch wenn in Abschnitt 8.1 alle Funktionen einer gitterbasierten Datenfusion auf die vier Basisfunktionen zurückgeführt worden sind, so ist es doch nicht notwendigerweise effizient, diese in jedem Fall einzeln aufzuspalten (Anforderung NF18). In Fällen, in denen dies nicht sinnvoll ist, werden die Algorithmen in der Komponente *Modifier* implementiert (siehe Abbildung 9.18). Eine *Modifier*-Komponente erhält Schreib- und Leserechte auf die *Layer*-Komponente. Darüber hinaus dürfen Instanzen der Komponente auch die Form und relative Lage der Gitterstruktur zum Anker verändern.

Auf diese Weise lassen sich z. B. zeitgesteuerte Funktionen wie das „Altern“ der Daten (siehe Abschnitt 8.1) effizient umsetzen. Ist der *Layer* ortsfest definiert, so ist i. d. R. eine Verschiebung notwendig, sobald der Versuchsträger sich zu nahe am Rand der beschriebenen Fläche bewegt.

Komponente: *View*

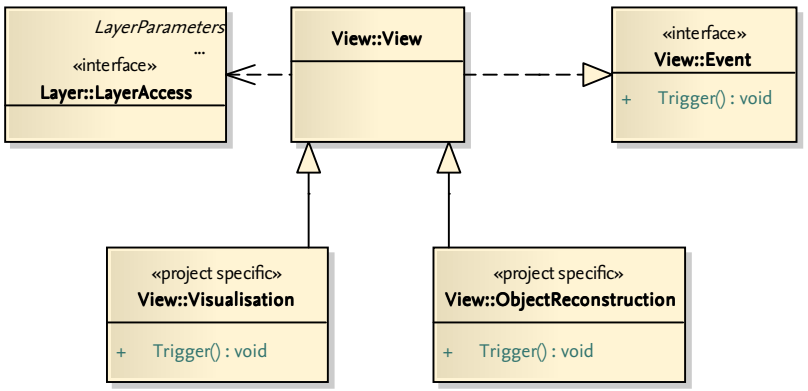


Abbildung 9.19: Klassendiagramm der Komponente *View*

Gitterdatenstrukturen haben einen sehr großen Speicherbedarf. Diese Datenmenge muss regelmäßig an die Applikation übertragen und schließlich von ihr ausgewertet werden. I. d. R. ist dies aufgrund von Bandbreitenbeschränkungen der eingesetzten Kommunikationstechnologien nicht möglich. Darüber hinaus wird in Anforderung NF18 eine effiziente Übertragungsform gefordert. Aus diesem Grund werden die Auswertungsalgorithmen einer Applikation innerhalb der *View*-Komponente implementiert und nur noch applikationsspezifische Datenstrukturen übermittelt. Die Komponenten stellen damit eine spezielle applikationsspezifische Sicht auf die Gitterdaten bereit.

Die *View*-Komponente erhält über die *LayerAccess*-Schnittstelle lesenden Zugriff auf die Gitterdatenstruktur (siehe Abbildung 9.19). Angestoßen wird die Auswertung des *Layers* über die *Event*-Schnittstelle. So kann z. B. zyklisch eine Objekthypothesenrekonstruktion durchgeführt werden oder eine Kopie der Gitterdaten zur Visualisierung erzeugt werden.

10 Softwarearchitektur - Laufzeitsicht

Die Laufzeitsicht einer Architektur beschreibt die Zusammenarbeit der Komponenten und Bausteine (Starke u. Hruschka, 2011, Seite 31). Während in der statischen Architektursicht Elemente nur einmal vorkommen und ihre Beziehung zu anderen Elementen dargestellt wird, bildet die Laufzeitsicht den Zustand des Systems im Einsatz ab. Je nach Konfiguration können mehrere Instanzen einer Komponente mit unterschiedlichen Konfigurationen und Aufgaben abgebildet werden. Da sich hieraus eine Vielzahl von möglichen Zusammenstellungen ergibt, werden i. d. R. nur exemplarische Abläufe von wichtigen Anwendungsfällen abgebildet sowie ihr Zusammenhang mit anderen Stakeholdern beschrieben.

Im Folgenden werden ausgewählte Abläufe der Komponenten und Klassen der Architektur beschrieben. Die Auswahl der Abläufe orientiert sich am Datenfluss vom Sensor hin zur Applikation. Die dabei eingesetzten Algorithmen stellen im Falle von abstrakten Architekturelementen mögliche Implementierungen dar. Für nicht abstrakte Elemente sind sie unabhängig von der späteren konkreten Realisierung der Architektur.

10.1 Sensor Specific Layer

Die Interpretation des Datenstroms eines Sensors ist meist sehr speziell an diesen angepasst. Deshalb wird an dieser Stelle nur auf einen fiktiven Sensor eingegangen. Dieser Sensor empfängt einen Nachrichtentyp, in dem bis zu n Objekthypothesen übertragen werden. Diese Objekthypothesen werden durch ein *PointModel* und ein *ConstantVelocityModel* beschrieben.

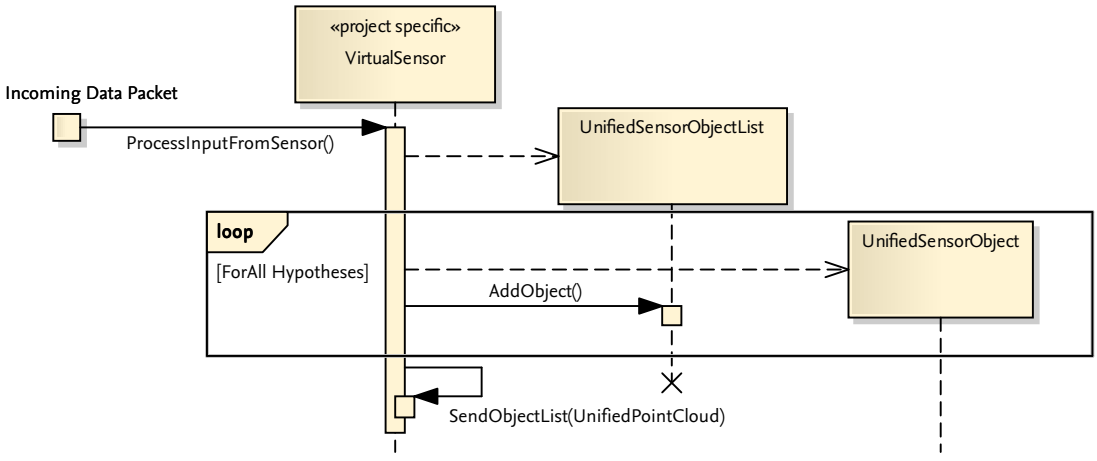


Abbildung 10.1: Sequenzdiagramm der Dekodierung des Protokolls eines fiktiven Sensors

Abbildung 10.1 beschreibt den Ablauf der Dekodierung des Protokolls des Sensors. Hierbei wird zunächst das Datenpaket empfangen und anschließend eine Instanz der Klasse *UnifiedSensorObjectList* angelegt. Diesem Objekt werden schließlich bis zu n Objekthypothesen hinzugefügt und abschließend über die Schnittstelle *UnifiedSensorObjectList* zur Verfügung gestellt.

10.2 Sensor Data Fusion Layer

Wie in Abschnitt 9 wird die objekthypothesenbasierte und gitterbasierte Fusion im Folgenden getrennt beschrieben. Die dargestellten Abläufe führen die Verarbeitung von Messdaten aus dem vorangegangenen Abschnitt fort.

10.2.1 Objekthypothesenbasierte Fusion

In einem objekthypothesenbasierten Fusionsmodul arbeiten unterschiedliche Klassen und Komponenten zusammen. Basierend auf der Annahme eines soeben gestarteten Fusionsmoduls werden die exemplarischen Abläufe dargestellt, welche sich durch den wiederholten Empfang von neuen Sensordaten ergeben. Da ein objekthypothesenbasiertes Fusionsmodul die logische Architektur aus Abschnitt 8 umsetzt, wird im Wesentlichen entlang des Datenflusses aus Abbildung 8.1 vorgegangen. Begonnen wird mit der Erzeugung eines neuen Tracks, gefolgt von der Verarbeitung neuer Messwerte und einer Darstellung eines möglichen Ablaufs des *TrackManagements*.

Erzeugung von neuen Tracks

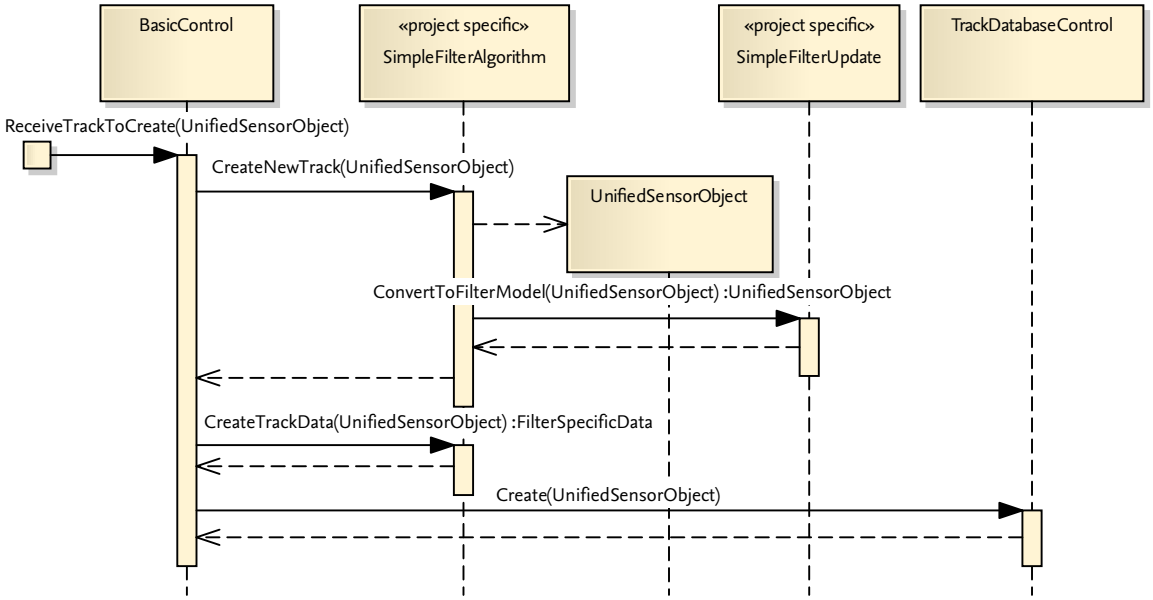


Abbildung 10.2: Sequenzdiagramm der Erzeugung eines neuen Tracks

Wird dem Fusionsmodul an seiner externen Schnittstelle eine Objekthypothese zur Erzeugung eines neuen Tracks übergeben, so wird diese durch die Klasse *BasicControl* der *Basis*-Komponente verarbeitet (siehe Abbildung 10.2). Sie nutzt die Schnittstelle *CreateNewTrack* der *Filter*-Komponente, um die Hypothese in die von der *Filter*-Komponente benötigte Beschreibung zu konvertieren. Nach der Umwandlung des Objekthypothesenmodells erhält die *Filter*-Komponente die Möglichkeit, dem Track interne Datenstrukturen zuzuordnen. Abschließend wird der neue Track in der Komponente *TrackDatabase* gespeichert und steht nun zur Zuordnung für neue Messdaten zur Verfügung.

Verarbeitung von neuen Messdaten

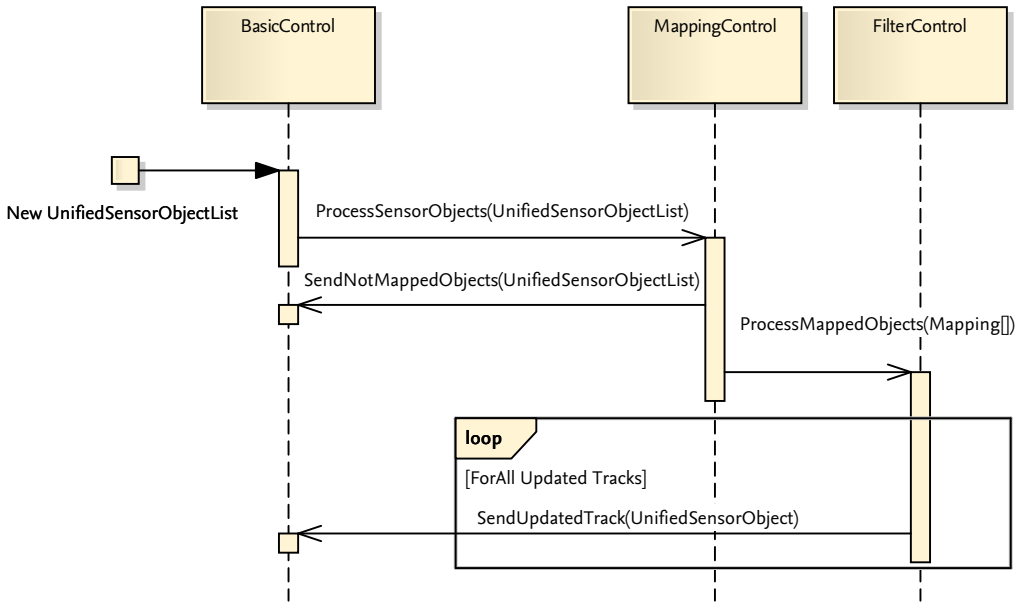


Abbildung 10.3: Sequenzdiagramm der Verarbeitung einer neuen Objekthypothesenliste

Die Abbildung 10.3 stellt die Verarbeitung von Messdaten durch die Komponenten *Basic*, *Mapping* und *Filter* sowie die Umsetzung des Fassade-Entwurfsmusters dar. In dem beschriebenen Fall wird die Objekthypothesenliste eines Sensors durch die *Basic*-Komponente empfangen. Diese schickt die Daten in einem asynchronen Aufruf an die *Mapping*-Komponente weiter. Nach erfolgter Zuordnung von bekannten Tracks zu den Objekthypothesen des Sensors werden die Daten an die *Filter*-Komponente gesendet. Kann eine Zuordnung nicht für alle Objekthypothesen erfolgen, so werden diese erneut an die *Basic*-Komponente zum Versand übergeben. Die *Filter*-Komponente fusioniert nun die Messdaten mit den zugeordneten Tracks und schickt anschließend für jeden aktualisierten Track eine Nachricht an die *Basic*-Komponente zum Versand an die Applikation.

Abbildung 10.4 stellt den Ablauf innerhalb der Komponente *Mapping* detaillierter dar. Nach dem Empfang einer neuen Objekthypothesenliste wird zunächst der Zugriff auf Tracks innerhalb der *TrackDatabase* gesichert. Anschließend werden alle Tracks kopiert und mithilfe der Klasse *FilterAlgorithm* auf den Messzeitpunkt der eingehenden Messdaten prädiziert. Nun kann der exklusive Zugriff auf den Datenbestand wieder aufgegeben werden. Darauf folgend wird der eigentliche *MappingAlgorithm* ausgeführt. Durch die abstrakte Definition und die damit verbundene Variabilität der Klasse wird hier kein konkreter Algorithmus beschrieben. Abschließend spaltet die *MappingControl*-Klasse die Ergebnismenge in zugeordnete und nicht zugeordnete Daten auf und übergibt diese einerseits der *Filter*- und andererseits der *Basic*-Komponente.

Nach dem Zuordnungsverfahren werden die zugeordneten Daten durch die *Filter*-Komponente weiter verarbeitet (siehe Abbildung 10.5). Die Menge aller Zuordnungen wird zunächst in individuelle Zuordnungen aufgespalten. Nun wird der exklusive Zugriff auf den entsprechenden Track in der *TrackDatabase* gesichert und dieser aus dem Datenbestand

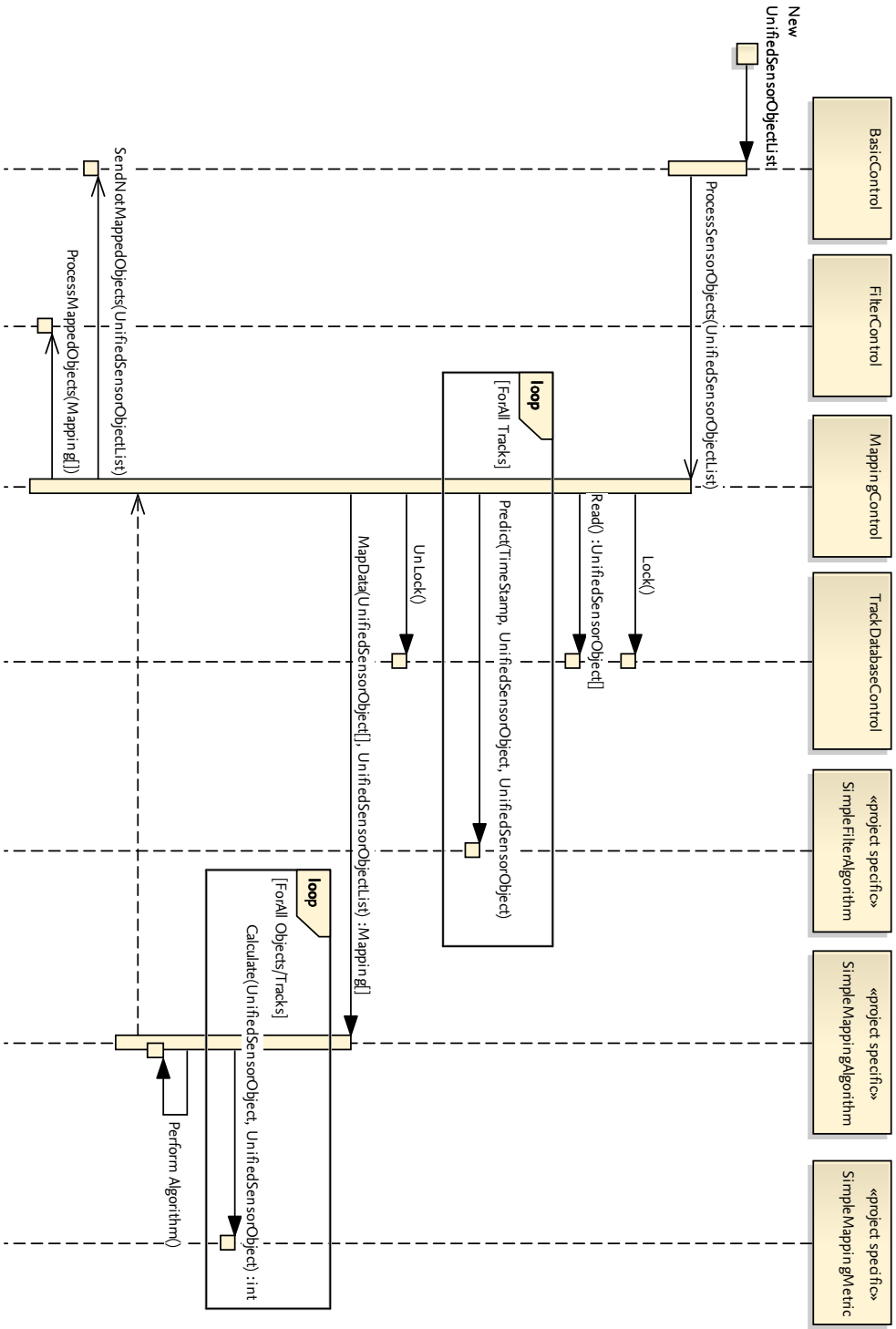


Abbildung 10.4: Sequenzdiagramm der Zuordnung von gemessenen Objekthypothesen zu bestehenden Tracks

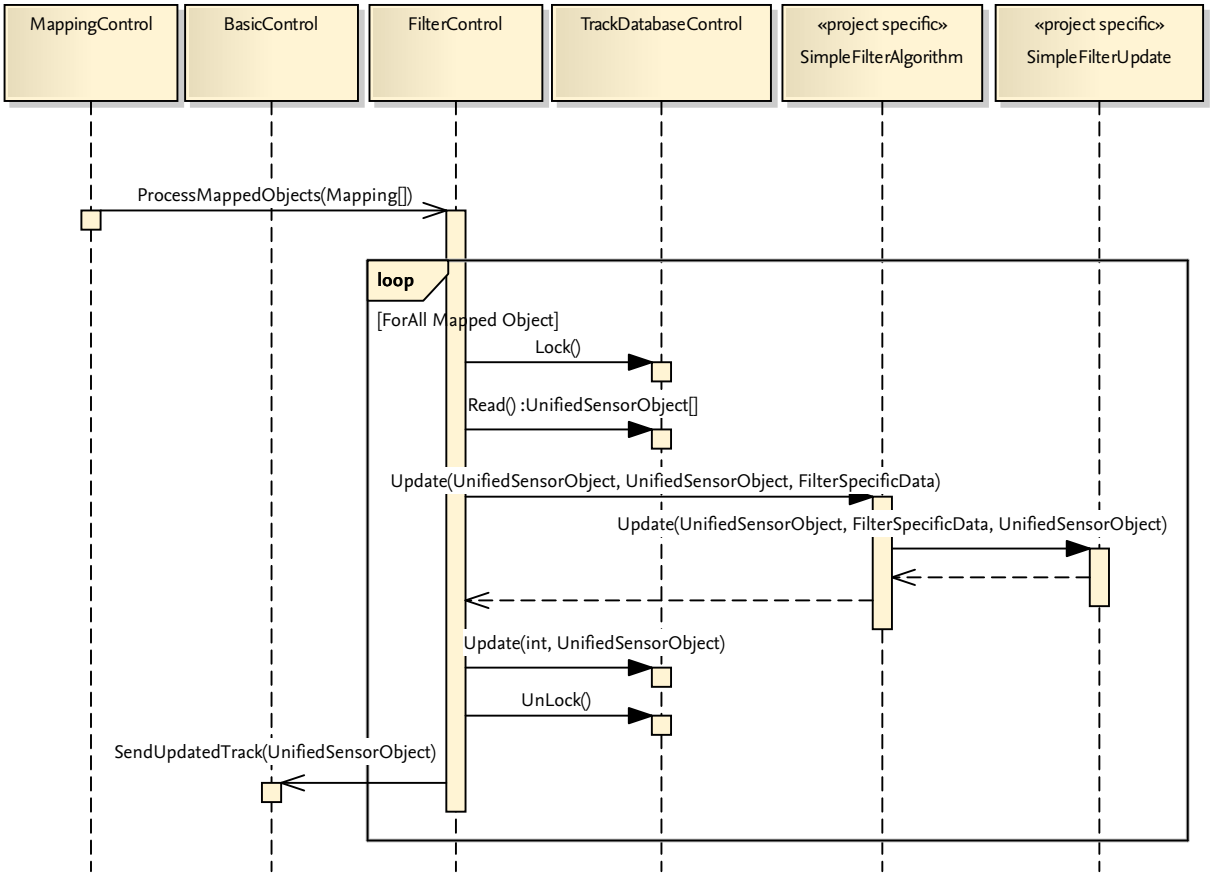


Abbildung 10.5: Sequenzdiagramm des Filterprozesses

ausgelesen. Dies ist nötig, da zwischen dem Zuordnungsverfahren und der *Filter*-Komponente evtl. andere Komponenten (z. B. das *TrackManagement* oder andere *Filter*-Instanzen) den Track verändert haben könnten. Anschließend übernimmt der *FilterAlgorithm* die Fusion von Track und Messdatum. Um das Messdatum auf die vom Filter benötigten Eingangsgrößen anzupassen, wird die Klasse *FilterUpdate* genutzt. Anhand der Typen des Geometrie- und Bewegungsmodells wird hier die entsprechende Konvertierungsvorschrift ausgewählt und schließlich die Fusion durchgeführt. Abschließend wird der Inhalt der *TrackDatabase* aktualisiert, der Zugriff freigeben und der Track an die *Basic*-Komponente überstellt.

Ablauf des *TrackManagements*

Das *TrackManagement* stellt eine Möglichkeit dar, zyklisch Berechnungen auf dem Datenbestand durchzuführen oder Tracks zu verändern. Die hier verwendeten Algorithmen sind überwiegend filterspezifisch. Im Folgenden wird deshalb nur sehr abstrakt auf den Ablauf eingegangen (siehe Abbildung 10.6). Nach Ablauf einer Zeitspanne wird zunächst eine allgemeine Pflege des Datenbestandes durchgeführt. Möglich wäre hier beispielsweise das Entfernen von Tracks, die eine gewisse Zeit keine Aktualisierung erfahren haben. Anschließend wird über die Schnittstelle *TrackManagementFilter* der Komponente *Filter* das filterspezifische *TrackManagement* durchgeführt.

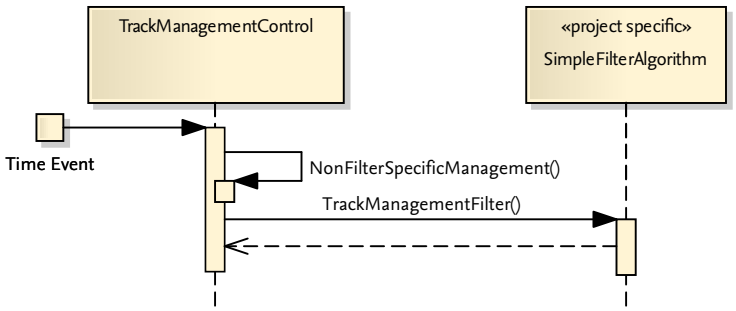


Abbildung 10.6: Sequenzdiagramm des TrackManagements

10.2.2 Gitterbasierte Fusion

Auch für gitterbasierte Fusionsmodule wird die Verarbeitung von eingehenden Messdaten sowie deren Auswertung exemplarisch beschrieben. Da ein gitterbasiertes Fusionsmodul die logische Architektur aus Abschnitt 8 umsetzt, wird im Folgenden entlang des Datenflusses aus Abbildung 8.1 vorgegangen. Da die Verarbeitung von Sensordaten eines Lasersensors ein häufiger Anwendungsfall für eine gitterbasierte Fusion ist, wird für die folgenden Abläufe auf diesen zurückgegriffen.

Verarbeitung von neuen Messdaten

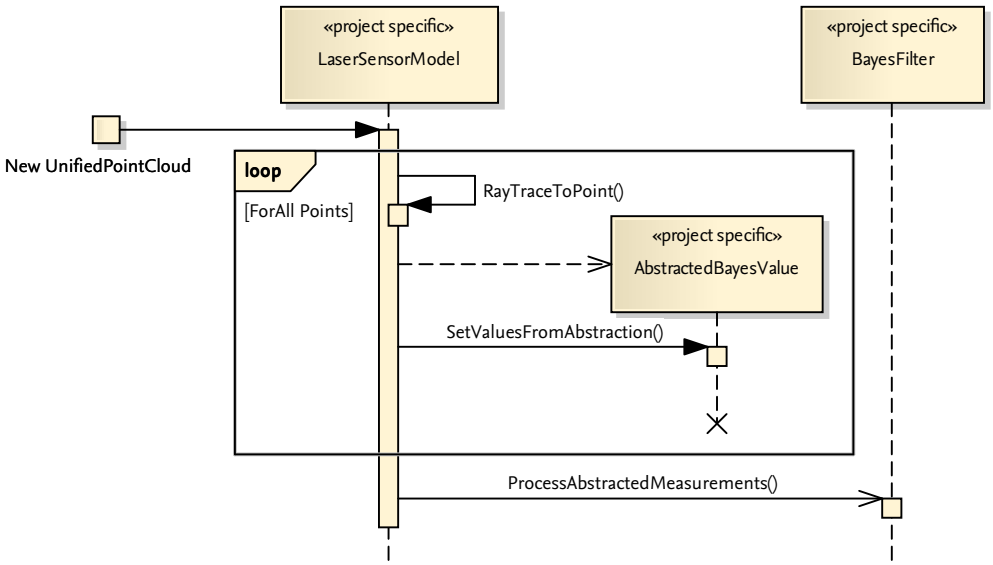


Abbildung 10.7: Sequenzdiagramm der Verarbeitung eines neuen Messdatums

Die erste Komponente, die an der Verarbeitung eines neuen Messdatums in einem gitterbasierten Fusionsmodul beteiligt ist, ist die *SensorModel*-Komponente. In Abbildung 10.7 ist die Verarbeitung eines neuen Messwerts eines Lasersensors dargestellt. Je nach Modellierung des Sensormessverhaltens kann die Verarbeitung auch anders gestaltet werden. In diesem Fall

bestimmt das Modell alle Zellen eines *Layers*, die von dem Weg des Laserstrahls gekreuzt werden. Anschließend wird für jede dieser Zellen eine Instanz der *AbstractedBayesValue*-Klasse erzeugt und der Wert entsprechend einer Abstraktionsvorschrift gesetzt. Die Liste mit allen erzeugten *AbstractedBayesValues* wird schlussendlich an die *Filter*-Komponente übergeben.

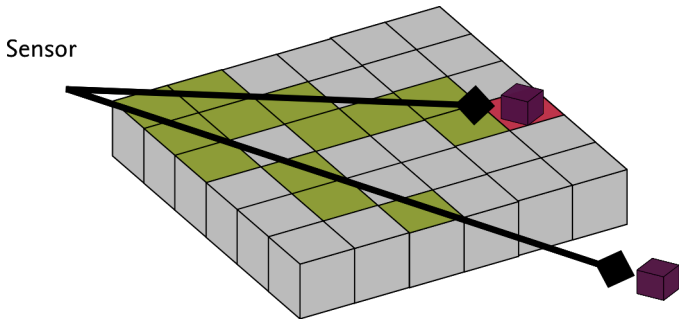


Abbildung 10.8: Sensormodell am Beispiel eines Lasersensors

Abbildung 10.8 zeigt ein mögliches Sensormodell eines Lasersensors. Hierbei wird der direkte Weg vom Ursprung des Sensors zum Reflektionspunkt mit dem Bresenham-Algorithmus (Bresenham, 1965) diskretisiert. Da der Sensorursprung sich außerhalb des *Layers* befindet, kann die erste Zelle nicht berücksichtigt werden. Darauf folgende Zellen werden der Ergebnismenge hinzugefügt und bis zum Reflektionspunkt mit einem Wert für „freie Fläche“ annotiert. Die Zelle, in der der Reflektionspunkt liegt, wird mit einem Wert für „belegte Fläche“ annotiert. Bei dem zweiten Messdatum liegen sowohl Ursprung als auch Reflektionspunkt außerhalb der Gitterstruktur. Hierbei werden nur Zellen in der Schnittmenge aus Gitterstruktur und Strahlengang der Ergebnismenge hinzugefügt.

Die Verarbeitung von abstrahierten Messwerten in einer *Filter*-Komponente beginnt mit der Sortierung der eingehenden Liste nach *GridBlocks* (siehe Abbildung 10.9). Diese Sortierung ist zwar nicht unbedingt notwendig, sollte aber durchgeführt werden, um unnötiges Anfordern von exklusiven Zugriffsrechten auf *GridBlocks* zu vermeiden. Anschließend werden die erzeugten Messwertgruppen verarbeitet. Zunächst wird der aktuelle *GridBlock* angefordert und der exklusive Zugriff darauf gesichert. Nun können die einzelnen *AbstractedBayesValues* anhand der Kombinationsregel von Bayes mit den aktuellen Werten der Zelle fusioniert werden.

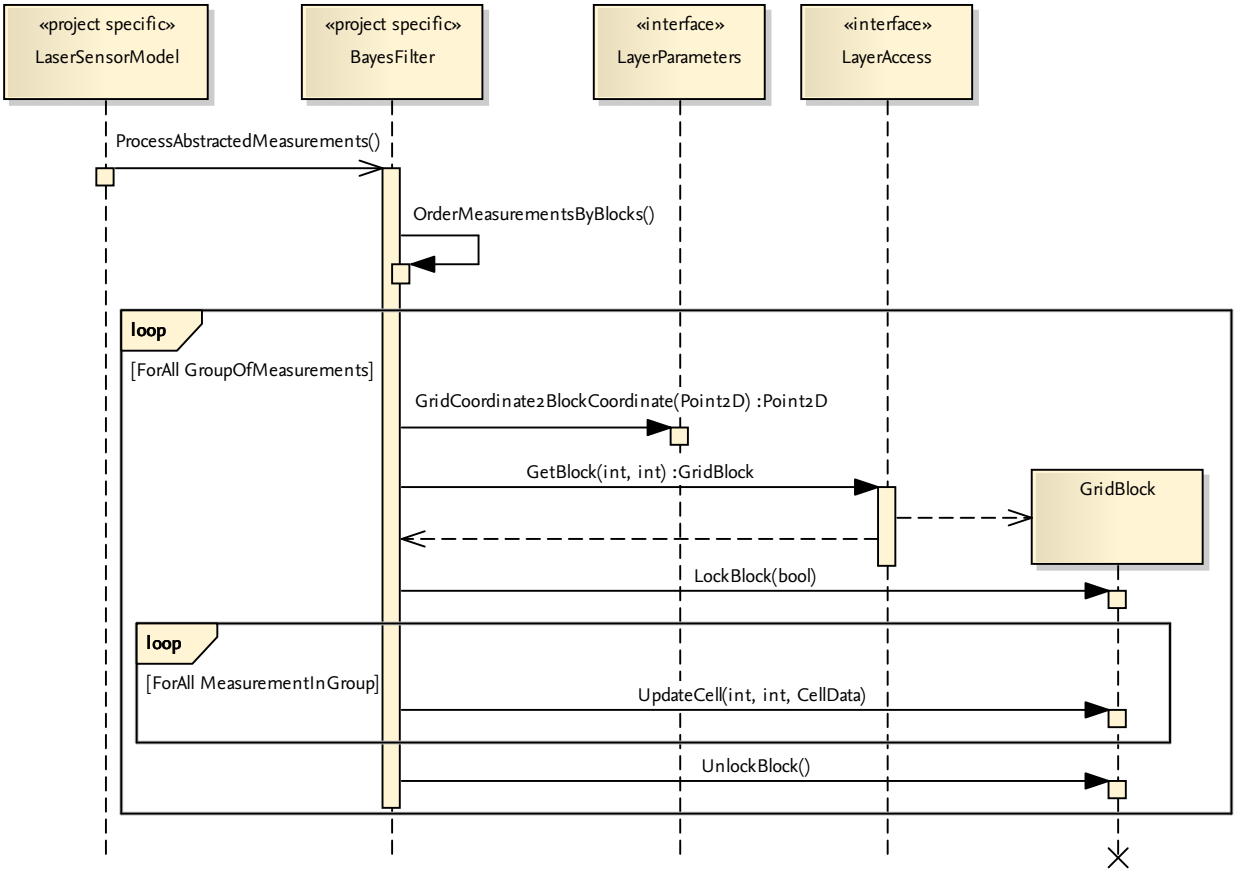


Abbildung 10.9: Sequenzdiagramm Fusion von abstrahierten Werten mit den Zellen eines Layers

Ereignis gesteuerte Komponenten

Die zuvor verarbeiteten Messdaten werden durch den Algorithmus in Abbildung 10.10 einem Anwender präsentiert. Hierzu wird aufgrund eines Ereignisses (z. B. zeitgesteuert) der Algorithmus einer *View*-Komponente gestartet. Dieser bestimmt die Kerndaten des *Layers* über die Schnittstelle *LayerParameters*. Anschließend iteriert der Algorithmus über alle *GridBlocks* sowie alle enthaltenen Zellen und präsentiert diese dem Nutzer.

I. d. R. werden nicht alle Zellen eines *Layers* regelmäßig aktualisiert. Deshalb soll die Unsicherheit der Daten aufgrund des Alters in diesem Beispiel durch eine Veränderung der Werte einer Zelle abgebildet werden. Im Falle eines Layers mit Bayes-Zellen könnte eine solche Funktion durch eine Verschiebung des Wahrscheinlichkeitswertes hin zu 50% abgebildet werden. In Abbildung 10.11 ist eine Implementierung der Komponente *Modifier* dargestellt. Sie bestimmt zunächst einmal die Ausmaße des Layers und iteriert anschließend über alle *GridBlocks* und Zellen, um diese Funktion anzuwenden. Da alle Zellen gleichzeitig „altern“, konvergieren sie gemeinsam gegen den Wert für „unentschieden“. Bei Zellen, die regelmäßig aktualisiert werden, bildet die Aktualisierung durch neue Messdaten jedoch ein Gegengewicht.

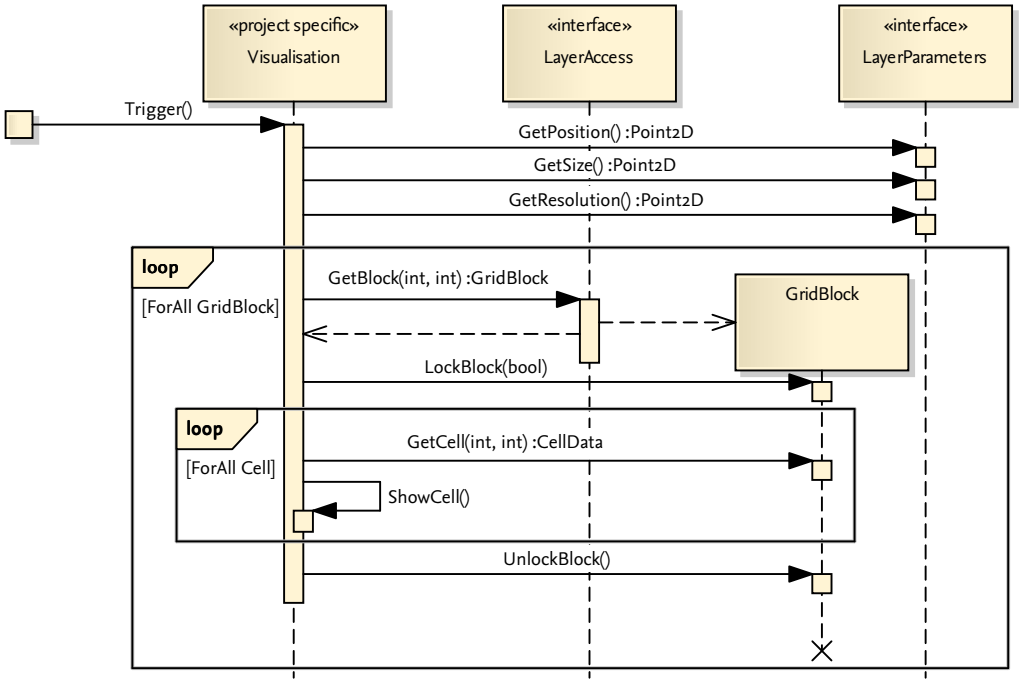


Abbildung 10.10: Sequenzdiagramm zur Präsentation des Inhalts eines Layers

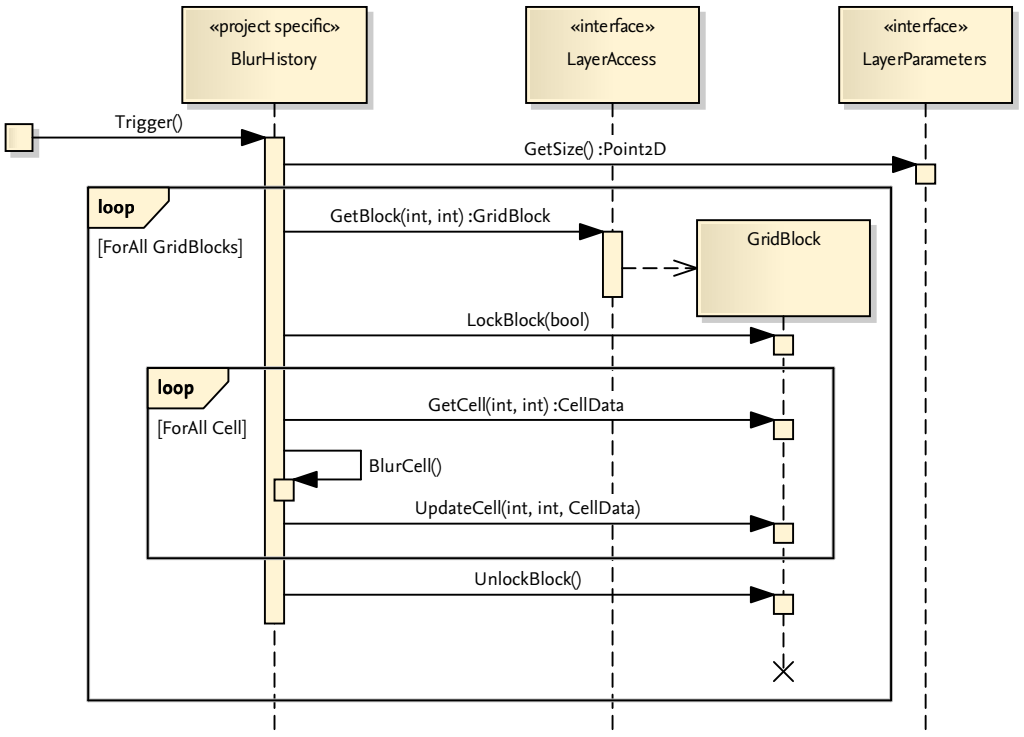


Abbildung 10.11: Sequenzdiagramm des „Alterns“ der Daten eines Layers

11 Evaluation der allgemeinen Anforderungen

Nach der Vorstellung der Architektur steht im Folgenden die Bewertung der Umsetzung im Vordergrund. Basierend auf den Anforderungen aus Abschnitt 7 werden die präsentierten Algorithmen evaluiert. Hierzu werden einerseits Wahr-Unwahr-Entscheidungen und andererseits simulierte Messdaten als Vergleich herangezogen. Zur Evaluation von Anforderungen werden dabei zwei Methoden eingesetzt:

Wahr-Unwahr-Entscheidungen: Die Erfüllung einiger der aufgestellten Anforderungen (z. B. Anforderung RB21) lässt sich eindeutig bewerten. Hierzu wird kurz beschrieben, durch welche Entscheidungen und Module die Erfüllung der Anforderungen gewährleistet wird.

Auswertung von Messdaten: Leistungsparameter des Umfeldwahrnehmungssystems können meist nicht durch Wahr-Unwahr-Entscheidungen anhand der Datenstrukturen und Algorithmen entschieden werden. Hierzu wurden mehrere Messungen durchgeführt und ausgewertet. Die Ergebnisse belegen die Erfüllung der Anforderungen.

Unterstützung unterschiedlicher Sensorkonfigurationen

In Anforderung FA1 wurde gefordert, dass unterschiedliche Sensorkonfigurationen unterstützt werden. Durch die Abstraktion der Modellbeschreibungen in den Datenstrukturen *UnifiedSensorObjectList* und *UnifiedPointCloud* wurde eine Festlegung auf einen speziellen Sensortyp vermieden. Darüber hinaus unterstützen die Strukturen des *SensorSpecificLayers* die Anbindung neuer Sensoren sowie die Wiederverwertung von bereits bestehenden Sensoranbindungen in anderen Projekten (siehe Abschnitt 9.1).

Unabhängigkeit von Basisfunktionen und Algorithmen und Abbildung der logischen Architektur einer Fusionsstruktur

Die Strukturen der Fusionsmodule ermöglichen durch die Separation der einzelnen Komponenten und deren weitere Unterteilung in Klassen eine weitgehende Unabhängigkeit der Algorithmen voneinander (Anforderung NF13). Hierdurch lassen sich einzelne Elemente einfach wiederverwenden. So nutzen beispielsweise das *MainFusionModule* und das *PreTrackingFusionModule* aufeinander aufbauende Zuordnungsalgorithmen (siehe Abschnitt 14.2.2), jedoch mit spezifischen Metriken. Darüber hinaus wurden die in Abschnitt 8 beschriebenen vier Elementarfunktionen in den Komponenten und Klassen für gitter- und objekthypothesenbasierte Datenfusionen abgebildet (Anforderung FA2).

Einsatz in unterschiedlichen Projekten (Anforderung NF14)

Die mithilfe der Architektur zur Umfeldwahrnehmung umgesetzten Softwarekomponenten wurde in unterschiedlichen Projekten für Fahrten auf Testgeländen genutzt. Als Hauptnutzer trat das Projekt Stadtpilot auf (siehe Abschnitt 3). Der Versuchsträger Leonie wurde jedoch auch für das Projekt Koline (Saust u. a., 2010) eingesetzt. Darüber hinaus wurden Komponenten in den Lehrveranstaltungen des Instituts für Regelungstechnik zur Auswertung von Infrarot- und Ultraschallsensoren angewendet.

Einsatz ohne Sensordatenfusion

Im Projekt Stadtpilot werden die Daten des Sensors Hella IDIS 2 direkt durch die Applikation genutzt. Innerhalb der Architektur werden nur die Komponenten des *Sensor Specific Layers* genutzt, um das Sensorprotokoll zu dekodieren. Der Einsatz des Fusionsmoduls *NoOpFusionModule* ermöglicht die Weiterleitung dieser Daten durch den *Sensor Data Fusion Layers* an die Applikation (Anforderung FA3).

Gitter- und objekthypothesenbasierte Fusion

In den Abschnitten 9.2.1 und 9.2.2 wird ausführlich auf die Strukturen zur Umsetzung von gitter- und objekthypothesenbasierten Fusionen eingegangen. Darauf folgend wird in Teil III die Umsetzung der Sensordatenfusion im Projekt Stadtpilot beschrieben (Anforderung FA4 und FA5).

Flexibles Objekthypothesenmodell

Um die Verarbeitung von Sensordaten zu vereinfachen, ist ein einheitliches Datenformat zur Beschreibung der Objekthypothesen von Vorteil. Würde diese Beschreibung sich auf ein Geometrie- und Bewegungsmodell festlegen, so wäre es nicht möglich, die Eigenschaften der Daten aller Sensoren abzubilden. Hierzu wurde zunächst das Objekthypothesenmodell in ein Bewegungs- und ein Geometriemodell aufgespalten. Durch die Kombination von unterschiedlichen Ableitungen der Basisklassen *MovementModel* und *GeometricModel* können so neue Modelle erstellt oder bestehende flexibel erweitert werden (Anforderung NF15).

UnifiedSensorObjectList

Die Datenstruktur *UnifiedSensorObjectList* enthält eine ganze Reihe von geforderten Daten. So enthält sie die aktuelle Sensorkonfiguration (Anforderung FA6) sowie eine Liste von einzelnen Objekthypothesen. Jede dieser Hypothesen ist gekennzeichnet durch eine Identifikationsnummer (Anforderung FA9) und wird annotiert durch das Alter der Daten (Anforderung FA7), den Erstellungszeitpunkt (Anforderung FA8) sowie eine getrennte Modellbeschreibung für die Geometrie- und Bewegungseigenschaften (Anforderung NF15).

Keine Festlegung von Fusionsalgorithmen durch die Architektur

Die Schnittstellen zwischen den einzelnen Komponenten der Fusionsmodule beschränken sich darauf, den Datenfluss festzulegen. Die eigentliche Datenverarbeitung wird nicht durch die Architektur festgelegt (Anforderung NF16). Darüber hinaus ermöglicht die Architektur, filterspezifische Datenstrukturen für Objekthypothesen zusätzlich abzulegen und so weitere Daten zu speichern.

Kombination von Fusionsstrukturen

Durch die Vereinheitlichung der Ein- und Ausgangsschnittstellen der Fusionsmodule lassen sich diese, je nach Anforderungen des Projekts, frei miteinander kombinieren. In Abschnitt 9.2 wird dies am Beispiel einer hierarchischen Struktur verschiedener Module gezeigt (Anforderung FA10).

Unterschiedliche Software- und Hardware-Schnittstellen

Abschnitt 9.1 führt die Datenstruktur *UnifiedDataPacket* ein. In dieser Datenstruktur werden die Binärdaten eines Datenpakets abgelegt und zusammen mit der Information über die Herkunft an die *SensorDataDeEncoder*-Komponente übertragen. Diese Vereinheitlichung des Datenstroms ermöglicht es, gleiche Informationen, die über unterschiedliche Datenbusse bereitgestellt werden, auf die gleiche Weise zu verarbeiten (Anforderung FA11).

Aufzeichnung von Sensor- und Fusionsdaten

Im Projekt Stadtpilot wurde ein System zur Aufzeichnung von über das Kommunikationsframework RTI DDS verschickten Daten entwickelt (Schütt, 2011). Die Datenstrukturen *UnifiedPointCloud*, *UnifiedSensorObjectList* und *UnifiedDataPacket* wurden so entwickelt, dass sie über das RTI DDS verschickt werden können. Damit ist eine Aufzeichnung sowohl der Sensordaten als auch der Ergebnisse der Fusionsmodule möglich (Anforderung FA12).

Durch die Fusionsarchitektur bedingte Latenzen

Um eine Messung der durch die Architektur bedingten Latenzen durchzuführen, ist eine konkrete Implementierung notwendig. Hierzu wurde die Umsetzung im Projekt Stadtpilot herangezogen. Abbildung 11.1 zeigt eine Latenzmessung bei der Verarbeitung von Objekthypothesen durch ein Fusionsmodul, das ausschließlich die minimalen Anforderungen an die entsprechenden Komponenten umsetzt (*NoOpFusionModule*). Hierbei wird auf den Einsatz von Filter- und Zuordnungsalgorithmen vollständig verzichtet. Es werden lediglich die notwendigen Speicherkopien zur Umsetzung des Datenflusses durchgeführt. Dargestellt wird die Latenz über der Anzahl der derzeitig verwalteten Anzahl an Objekthypothesen. Zu sehen ist, dass die untere Schranke des 95%-Konfidenzintervalls auch bei 150 Tracks bei nur knapp über $2ms$ liegt[1] und mit der Anzahl der Objekthypothesen linear ansteigt

[1] Intel Core i7 2640M, 2,8 GHz, 4 GB Speicher, Linux 3.2.0, ADTF 2.7.0

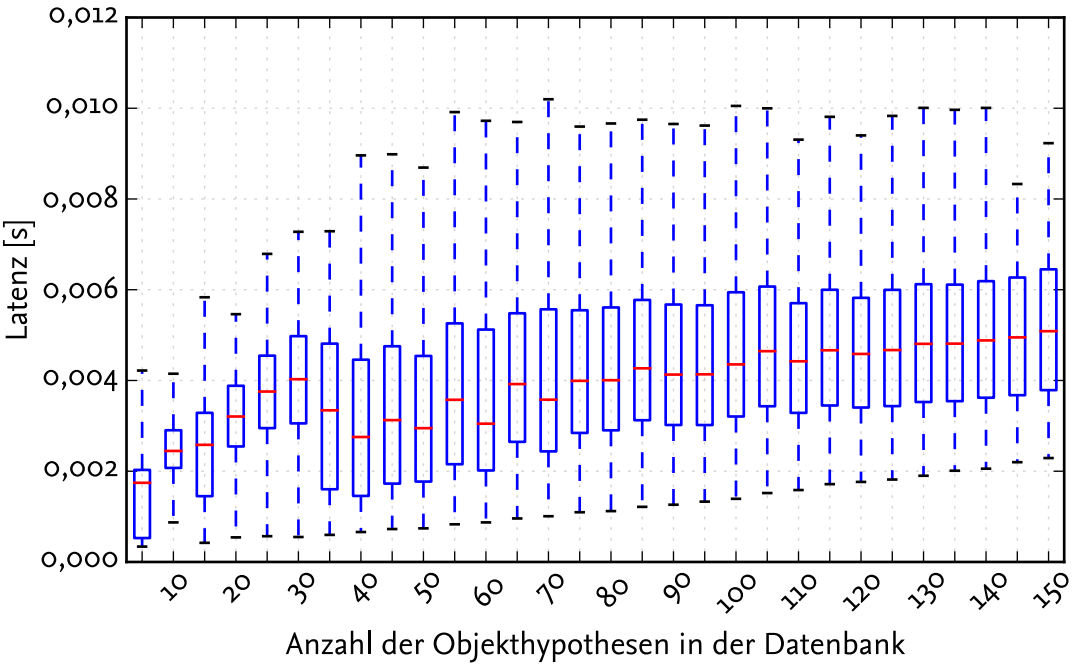


Abbildung 11.1: Durch Software-Architektur bedingte Latenz bei der Verarbeitung von Objekthypothesen (Box: mittleres Quartil, Whisker: 95%-Intervall)

(Anforderung NF17). Die Werte für geringe Mengen an Tracks sind vor allem durch das Rauschen der Aufgabenplanung des ausführenden Systems geprägt. Auch die Konstanz der oberen Schranke des Konfidenzintervalls lässt sich auf das Hostsystem zurückführen.

Wiederverwendungsgrad

Durch die Implementierung der Elementarfunktionen in Komponenten und Klassen sowie durch die Abstraktion des Objekthypothesenmodells lassen sich die implementierten Algorithmen neu kombinieren. Die Möglichkeit, diese Algorithmen in unterschiedlichen Fusionsmodulen zu nutzen, ist damit gegeben (Anforderung NF13 und NF14).

Effizienter Zugriff auf Gitterdaten

Eine Applikation erhält keinen direkten Zugriff auf die Gitterdaten einer *Layer*-Komponente. Hierzu werden *View*-Komponenten genutzt (siehe Abschnitt 9.2.2). Diese können direkt auf den Speicher des *Layers* zugreifen und durch Kondensierung der Daten einen effizienten Zugriff der Applikation auf die benötigten Größen sicherstellen (Anforderung NF18).

Randbedingungen

Umgesetzt wurde die Architektur zur Umfeldwahrnehmung innerhalb des Projekts Stadtpilot. Dabei wurden die projektspezifischen Softwareprodukte genutzt. Die Architektur wurde im Entwicklungsframework ADTF in der Programmiersprache C++ umgesetzt (Anforderungen RB19 und RB21). Zur Kommunikation über Prozessgrenzen hinweg wurde das Kommunikationsframework RTI DDS eingesetzt (Anforderung RB20). Basierend auf den Zielen des Projekts Stadtpilot wurde die Umsetzung der Architektur auf die Bedürfnisse der „Automotive"-Domäne hinsichtlich üblicher Sensoren und Schnittstellen ausgerichtet (Anforderung RB22).

Übersicht über die bisher erfüllten Anforderungen

Die vorangegangenen Absätze lassen sich in folgender Tabelle zusammenfassen:

Anforderung	Erfüllt		Anforderung	Erfüllt
FA1	✓		FA12	✓
FA2	✓		NF13	✓
FA3	✓		NF14	✓
FA4	✓		NF15	✓
FA5	✓		NF16	✓
FA6	✓		NF17	✓
FA7	✓		NF18	✓
FA8	✓		RB19	✓
FA9	✓		RB20	✓
FA10	✓		RB21	✓
FA11	✓		RB22	✓

11.1 Grenzen der Softwarearchitektur

Die in dieser Arbeit beschriebene Umsetzung der allgemeinen Anforderungen in eine Softwarearchitektur zur Umfeldwahrnehmung bildet die notwendigen Strukturen zur Erstellung von Sensordatenfusionen im Fahrzeug ab. Neben der hier beschriebenen strengen Trennung von Sensoren und Fusionsprozess gibt es auch eine andere Herangehensweise. Dabei werden Sensoren und Fusionsprozess stärker kombiniert und vor allem sensorspezifische Attribute der Messdaten (z. B. Klassifikationen oder Reflexionsstärken) mit in den Fusionsprozess einbezogen. Durch die einheitlichen Datenstrukturen *UnifiedPointCloud* und *UnifiedSensorObjectList* ist dies zunächst nicht offensichtlich möglich. Durch Ableitung einer der beiden Sensormodelle lassen sich diese Daten, die entweder dem Bewegungs- oder Geometriemodell zugeordnet werden können, jedoch bis in ein Fusionsmodul leiten und anschließend dort verarbeiten. Sollen eine Vielzahl von unterschiedlichen sensorspezifischen Daten übermittelt werden, so ist zu überlegen, ob eine Erweiterung der *UnifiedSensorObjectList*-Datenstruktur um ein Binärfeld zur Kodierung der sensorspezifischen Daten sinnvoll erscheint.

Nach der Veröffentlichung der Arbeit von Mählisch zur Existenzschätzung von Objekthypothesen (Mählisch, 2009) setzt sich dieses Konzept immer weiter durch. Je nachdem, ob ein Messdatum des Sensors zur Existenzwahrscheinlichkeit einer Objekthypothese vorliegt

oder nicht, wird diese Information erst im Fusionsmodul berechnet oder muss bereits vom Sensor an dieses übermittelt werden. Durch die Beschränkung der Datenstruktur *UnifiedSensorObjectList* auf zwei Beschreibungsmodelle für die Geometrie und die Bewegung einer Objekthypothese lässt sich diese Information nicht ohne Weiteres an eine Applikation übertragen. Zur Umsetzung dieses Fusionskonzepts bietet sich die Erweiterung der Datenstruktur um ein weiteres Beschreibungsmodell für die Existenzwahrscheinlichkeit an.

12 Zusammenfassung

Im Teil über die Softwarearchitektur der Fusion von Umfeld wahrnehmenden Sensoren wurde zunächst der Zweck der beschriebenen Architektur zur Fusion von Umfeld wahrnehmenden Sensoren näher definiert. Anschließend wurde allgemein auf Softwarearchitektur, ihre Definition und ihre Ziele eingegangen. Beschrieben wurden gitter- und objekthypothesenbasierte Fusionen.

Die Festlegung des Architekturkontexts ermöglichte die Identifizierung von Stakeholdern zur Bestimmung von Anforderungen an die Architektur sowie die Abgrenzung gegenüber anderen verwandten Anwendungsdomänen. Im Gegensatz zu anderen Implementierungen und Architekturen richtet sie sich ebenfalls nicht an die Fusion von Entscheidungen und auch die Klassifikation von Objekthypothesen wird nicht von ihr behandelt. Neben dem eigentlichen Abnehmer der fusionierten Messdaten treten auch die Entwickler sowie die Systemadministratoren als Stakeholder auf, sodass auch ihre Anforderungen durch die Architektur repräsentiert werden.

Zur Realisierung einer Softwarearchitektur ist ein genaues Verständnis der Arbeitsdomäne notwendig. Hierbei wurden innerhalb eines Fusionsprozesses vier Grundelemente identifiziert, die sich in allen Sensordatenfusionen wieder finden. Zunächst werden Tracks aus dem Datenbestand der Fusion anhand der eingehenden Messdaten selektiert (*Selektion*), anschließend auf die Eingangsgrößen des Filters abgebildet (*Abstraktion*) und schließlich mithilfe einer Kombinationsvorschrift miteinander fusioniert (*Fusion*). Ergänzt wird der Prozess durch eine *Datenspeicherung*. Werden als Messdaten auch zeitliche Ereignisse sowie leere Mengen für Messdaten und Tracks zugelassen, so ist damit der Aufbau von beliebigen Sensordatenfusionen möglich.

Die Anforderungen der Stakeholder bilden die Grundlage zur Entwicklung der Softwarearchitektur. Dabei wurden die Anforderungen in die drei Bereiche nicht-funktionale Anforderungen, funktionale Anforderungen und Randbedingungen aufgeteilt und nach ihrer Herkunft klassifiziert.

Die Softwarearchitektur teilt sich in die Ebenen *Sensor Specific Layer*, *Sensor Data Fusion Layer* und *Application* auf. Jede der Ebenen wurde aus der Baustein- sowie der Laufzeitsicht betrachtet. Der *Sensor Specific Layer* widmet sich der Anbindung von Sensoren an eine Sensordatenfusion. Dabei werden die sensorspezifischen Protokolle in zwei einheitliche Datenformate (*UnifiedSensorObjectList* bzw. *UnifiedPointCloud*) umgewandelt. Im *Sensor Data Fusion Layer* findet die eigentliche Fusion von Messdaten statt. Innerhalb der Ebene können mehrere Sensordatenfusionen, sogenannte Fusionsmodule, existieren. Jedes dieser Module stellt eine gitter- oder objekthypothesenbasierte Sensordatenfusion basierend auf der logischen Architektur einer Fusionsstruktur dar. Die Ausgangsdaten eines Fusionsmoduls können wiederum als Eingangsgröße für andere Fusionsmodule dienen und ermöglichen somit auch verteilte Fusionsarchitekturen.

Abgeschlossen wurde der Teil mit einer Auswertung zur Erfüllung der zuvor aufgestellten Anforderungen. Diese wurden anhand von Wahr-Unwahr-Entscheidungen und ausgewerteten Messdaten diskutiert. Alle Anforderungen konnten als erfüllt angesehen werden.

Teil III: Realisierung der Architektur

Die in Teil II beschriebene Architektur stellt die Grundlage für ein real einsetzbares System zur Umfeldwahrnehmung bereit. Der nun folgende Teil beschreibt die projektspezifischen Anforderungen an die Realisierung der Architekturinstanz und die wesentlichen Algorithmen innerhalb des Umfeldwahrnehmungssystems des Projekts Stadtpilot. Dabei wurden sowohl objekthypothesen- als auch gitterbasierte Fusionsmodule umgesetzt.

13 Anforderungen aus dem Projekt Stadtpilot

Neben den in Abschnitt 7 aufgestellten Anforderungen sind zur Realisierung der Architektur weitere projektspezifische Anforderungen notwendig. Diese bilden die Grundlage für die Auswahl und das Design der Algorithmen in den Modulen der Architektur zur Fusion von Umfeld wahrnehmenden Sensoren. Im Folgenden werden diese nach dem gleichen Schema wie die Anforderungen an die Softwarearchitektur aufgestellt.

Zur Bestimmung der Anforderungen wurde vor allem auf die Funktionsdefinition aus Abschnitt 3.3 zurückgegriffen. Die Grundlage stellt im Folgenden das Szenario Herbst 2012 dar. Die geforderten Güten innerhalb der Anforderungen wurden auf einem gemeinsamen Workshop mit den am Projekt beteiligten Mitarbeitern festgelegt. Insbesondere die Abschätzung zur Latenz und zu anderen Größen kann in Anhang C nachvollzogen werden.

Funktionale Anforderungen aus dem Projekt Stadtpilot

SPFA23 Die Umfeldwahrnehmung soll möglichst einen $360°$ Blick um das Fahrzeug ermöglichen.

Der Versuchsträger besitzt während seines Betriebs im innerstädtischen Verkehr keine Vorzugsblickrichtung. Zwar ist der hauptsächlich auftretende Verkehr in Längsrichtung vor und hinter dem Versuchsträger anzusiedeln, jedoch muss das Fahrzeug ebenfalls Objekte aus Querrichtung wahrnehmen. Hierzu ist ein $360°$ Rundumblick notwendig.

SPFA24 Fahrzeuge im Geschwindigkeitsbereich von 0 bis $60\frac{km}{h}$ sollen wahrgenommen werden.

Im innerstädtischen Versuchsumfeld sind Geschwindigkeiten bis zu $50\frac{km}{h}$ erlaubt. Diese werden im realen Verkehr jedoch teilweise überschritten, sodass mit Verkehrsteilnehmern mit Geschwindigkeiten bis zu $60\frac{km}{h}$ zu rechnen ist. Neben sich bewegenden Verkehrsteilnehmern sind auch stehende Hindernisse in Form von abgestellten oder wartenden Fahrzeugen im Versuchsumfeld vorhanden.

SPFA25 Fußgänger und Fahrräder sollen als einzelne Objekthypothesen wahrgenommen werden.

Soweit die Auflösung und die Trennfähigkeit der eingesetzten Sensoren dies ermöglicht, sollen kleine Verkehrsteilnehmer (vor allem Fahrräder und Fußgänger) als einzelne Objekthypothesen wahrgenommen werden und nicht mit anderen Objekthypothesen kombiniert werden. Dies ermöglicht eine Behandlung dieser Objekte als Individuen.

SPFA26 Befahrbare und nicht befahrbare Bereiche der Straße müssen wahrgenommen werden.

Die in Abschnitt 3 erwähnte hochgenaue Straßenkarte dient als Grundlage für die Bahnplanung des Versuchsträgers. Da die Karte nicht die aktuelle Verkehrssituation widerspiegelt, muss sie durch eine Umfeldwahrnehmung ergänzt werden. Die Information über die aktuell durch den Versuchsträger befahrbaren bzw. nicht befahrbaren Bereiche der Karte können von der Bahnplanung/Regelung genutzt werden, um die Trajektorie des Fahrzeugs entsprechend anzupassen.

SPFA27 Die Geschwindigkeit der Objekthypothese eines Fremdfahrzeugs muss eine zum Betrieb einer komfortablen Folgefahrt ausreichende Güte aufweisen.

Eines der Hauptverhalten des Stadtpilot-Versuchsträgers ist das Folgen anderer Fahrzeuge. Hierzu muss die Geschwindigkeit des Egofahrzeugs dem des Folgefahrzeugs angepasst und ein sicherer Abstand eingehalten werden. Das Geschwindigkeitsdatum der Objekthypothese muss aus diesem Grund eine ausreichende Güte und Stabilität aufweisen, damit die Fahrzeuginsassen eine komfortable Fahrt erleben (siehe Abschnitt B).

SPFA28 Die Geschwindigkeit und Position der Objekthypothese eines anderen Fahrzeugs muss eine zur Entscheidung über einen Fahrstreifenwechsel ausreichende Güte aufweisen.

Neben der Folgefahrt sind Fahrstreifenwechsel in der Funktionsdefinition des Stadtpilot-Versuchsträgers enthalten. Zur Entscheidung, ob ein Fahrstreifenwechsel durchgeführt werden kann, wird der Abstand, die TTC sowie die Zeitlücke zwischen dem Ego- und Fremdfahrzeug von der Situationsanalyse genutzt (Ulbrich, 2011, Seite 83). Hierzu müssen das Geschwindigkeits- und Positionsdatum einer Objekthypothese eine ausreichende Güte und Stabilität aufweisen (siehe Abschnitt B).

SPFA29 Die Daten der Umfeldwahrnehmung müssen die Bestimmung der aktuellen Erfassungsbereiche der Sensoren zulassen.

Während des Abbiegens durch fließenden Verkehr oder während eines Fahrstreifenwechsels ist das Wissen über den aktuellen Erfassungsbereich der Sensorik von großer Bedeutung. Verdeckungen lassen sich beispielsweise auf andere Fahrzeuge oder Infrastruktur zurückführen. Die daraus resultierende Unsicherheit kann die Entscheidungsgrundlage für ein Manöver erheblich beeinflussen. Aus diesem Grund muss der aktuelle Erfassungsbereich der Sensoren bestimmbar sein.

SPFA30 Die Ausmaße anderer Fahrzeuge und der Randbebauung sollen wahrgenommen werden.

Im innerstädtischen Verkehr sind die Abstände zu anderen Verkehrsteilnehmern oft sehr gering. Zusätzlich ist mit einer erheblichen Menge von statischen Hindernissen zu rechnen, die in die Fahrbahn ragen können. Um dennoch eine sichere Bahnplanung durchführen zu können, müssen die Ausmaße anderer Verkehrsteilnehmer im Umfeld des Versuchsträgers wahrgenommen werden.

SPFA31 Stehende und bewegte Objekte sollen wahrgenommen werden.

Das innerstädtischen Umfeld besteht aus einer Vielzahl von Objekten: fahrende Fahrzeuge, Fußgänger, aber auch abgestellte Fahrzeuge und statische Infrastruktur gehören zum Bild des städtischen Verkehrs. Da viele dieser Hindernisse für das das Egofahrzeug relevant sind, sollen sie wahrgenommen werden.

SPFA32 Eine Objekthypothese stellt i. d. R. für das Egofahrzeug ein relevantes Objekt im Sinne eines Hindernisses oder eines anderen Verkehrsteilnehmers dar.

Objekthypothesen repräsentieren statische Hindernisse oder andere Verkehrsteilnehmer. Objekte, die dieser Klassifikation nicht standhalten oder beispielsweise aus Messfehlern eines Sensors resultieren, sind zu vermeiden.

Nicht-funktionale Anforderungen aus dem Projekt Stadtpilot

SPNF33 Die Verarbeitung der Sensordaten muss echtzeitfähig sein.

Der Stadtpilot-Versuchsträger bewegt sich im fließenden Verkehr und muss auf aktuelle Veränderungen des Verkehrsgeschehens unmittelbar reagieren können. Aus diesem Grund darf die Umfeldwahrnehmung die eingehenden Daten nicht langsamer verarbeiten als diese erzeugt werden.

SPNF34 Die Zustandsgrößen einer Objekthypothese sollen keine unrealistischen Daten enthalten.

Große Abweichungen der Zustandsgrößen einer Objekthypothese können in den verarbeitenden Applikationen zu erheblichen Reaktionen der Algorithmen führen. Aus diesem Grund dürfen nur Zustandsgrößen übermittelt werden, die sich innerhalb des erwarteten Bereichs der Zustandsgröße befinden.

Anforderung	Quelle:			Anforderung	Quelle:		
	System	Nutzer	Domäne		System	Nutzer	Domäne
SPFA23		✓		SPFA29		✓	
SPFA24		✓		SPFA30		✓	
SPFA25		✓		SPFA31		✓	
SPFA26		✓		SPFA32		✓	
SPFA27		✓		SPNF33		✓	
SPFA28		✓		SPNF34		✓	

Tabelle 13.1: Quellen der aufgestellten projektspezifischen Anforderungen

14 Objekthypothesenbasierte Fusion

Zur Umsetzung der Anforderungen des Stadtpilot-Projekts werden innerhalb des *Sensor Data Fusion Layers* drei objekthypothesenbasierte Fusionsmodule mit den aus Abschnitt 9 bekannten Elementen umgesetzt. Ergänzt werden die Fusionsmodule durch eine Komponente zur Reduzierung von Sensorgeistern, die die Erzeugung von neuen Tracks verhindern oder verzögern kann (siehe Abschnitt 14.3).

Abbildung 14.1 zeigt den Datenfluss sowie die Verteilung der Module auf die Ebenen der Softwarearchitektur. Durch die Fusionsmodule werden fünf Sensortypen verarbeitet. Die Daten eines weiteren Sensortyps, des Hella IDIS 2, werden über das *NoOpFusionModule* an die Applikation gesendet. Dieses Modul leitet die Daten ohne aufwändige Algorithmen weiter und ermöglicht durch die daraus resultierende kurze Latenz eine komfortable Folgefahrt.

Innerhalb des *Sensor Data Fusion Layers* befinden sich drei weitere Instanzen zweier unterschiedlicher Fusionsmodule.

Die Radarsensoren vom Typ SMS UMMR 2006 liefern in der eingesetzten Version sog. Ziellisten, d.h. die Ziele sind lediglich stabilisiert, jedoch nicht weiter vorverarbeitet. Aus diesem Grund ist zur Vorverarbeitung ein nachträgliches Tracking der Daten notwendig. Das *RadarFusionModule* ist eine speziell konfigurierte Instanz des *PreTrackingFusionModules* aus Abschnitt 14.2. Hierbei wurden die Parameter der *Mapping*-Komponente angepasst und eine zyklische Kopie der Trackliste erstellt, um einen virtuellen Sensor darzustellen.

Das *MainFusionModule* verarbeitet die Objekthypothesenlisten aller Sensoren sowie die Daten des *RadarFusionModules*. Als *Mapping*-Komponente kommt in diesem Modul der Algorithmus Minimum-Filter (Schneider, 2006, Seite 56f) mit einer euklidischen Metrik (Deza u. Deza, 2009, Seite 104) zum Einsatz. Zur Implementierung der *Filter*-Komponente wird das in Abschnitt 14.1 beschriebene konturklassifizierende Kalman-Filter mit variablem Objekthypothesenmodell genutzt.

Tracks, die innerhalb des *MainFusionModules* nicht durch die *Mapping*-Komponente zugeordnet werden können, werden durch das *PreTrackingFusionModule* verarbeitet (siehe Abschnitt 14.2). Dem Fusionsmodul liegt der Gedanke zugrunde, dass Objekthypothesen nur verfolgt werden können, wenn sie eine gewisse Konformität in Bezug auf das zugrunde liegende Modell aufweisen. Hierzu wurde eine *Mapping*-Komponente entwickelt, die ihre Gating-Parameter anhand der Anzahl neu erzeugter Tracks anpasst. Die Tracks dieses Fusionsmoduls werden mit einer *Filter*-Komponente verarbeitet, das einen Kalman-Filter umsetzt.

Wie aus der Abbildung 10.5 bekannt ist, stellt die *Filter*-Komponente jede erfolgte Aktualisierung eines Tracks über eine Schnittstelle bereit. Ist diese Schnittstelle direkt mit der Schnittstelle zur Erzeugung von neuen Tracks eines Fusionsmoduls gekoppelt (siehe Abbildung 10.2), so würden sie konvertiert und als neue Tracks dem Datenbestand des Moduls hinzugefügt werden. Durch eine entsprechende Untersuchung und Filterung des Datenstroms können neue Tracks verhindert oder verzögert werden. Dies geschieht in dem Modul *NewTrackSelektion*. Die Algorithmen für die Auswahl von neuen Tracks werden in Abschnitt 14.3 näher vorgestellt.

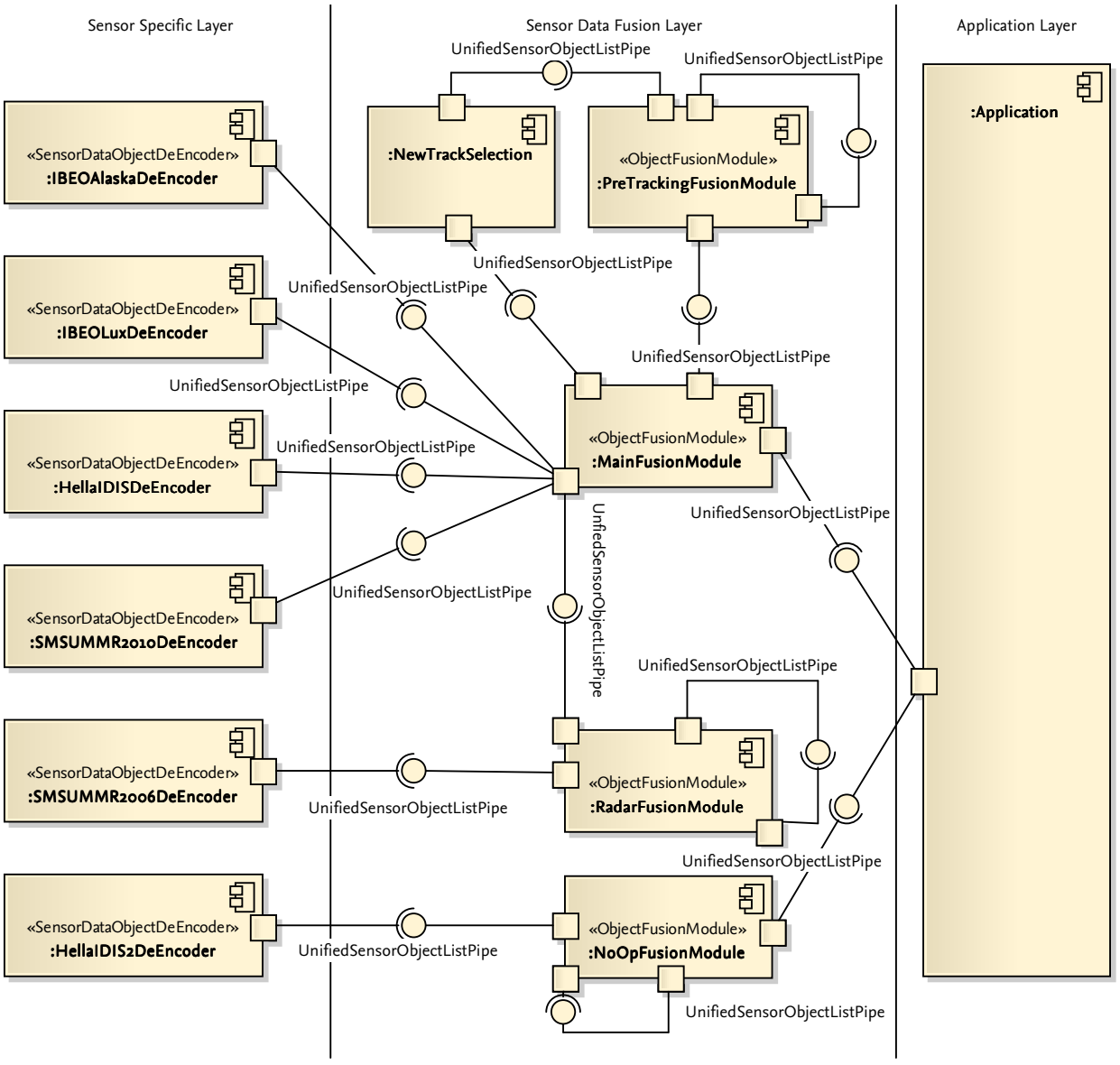


Abbildung 14.1: Datenfluss und Aufteilung der objekthypothesenbasierten Fusionsmodule im Projekt Stadtpilot. Die Typen der Schnittstellen zwischen den Komponenten finden sich in Abschnitt 9.2.1. Sie sind aus Gründen der Übersichtlichkeit nicht mit abgebildet.

14.1 Konturklassifizierender Kalman-Filter

Im städtischen Umfeld treten eine große Anzahl von Objekten mit unterschiedlichen Konturen auf. Eine Konzentration auf bestimmte Verkehrsteilnehmer und damit eine Einschränkung der Formen ist nicht ohne Weiteres möglich, da der Versuchsträger potentiell mit allen interagieren muss. Damit sind die klassischen Beschreibungen von Objekten durch Objekthypothesen auf Straßen höherer Ordnung nur bedingt einsetzbar. Aus diesem Grund wurde, aufbauend auf den Arbeiten von Effertz und Ohl, eine Beschreibung von Objekten im Fahrzeugumfeld durch Objekthypothesen mit offenem Polygonzug (Kontur) realisiert (Effertz, 2009; Ohl, 2007).

Im Vergleich zu den zitierten Vorgängerarbeiten hat sich das Versuchsumfeld jedoch deutlich erweitert. So war im Rahmen der DARPA Urban Challenge nur eine sehr geringe Menge PKW-Verkehr innerhalb des Versuchsfelds vorhanden. Auch lagen die real erreichten Geschwindigkeiten deutlich niedriger als im Umfeld des Braunschweiger Stadtrings. Darüber hinaus werden durch die von Effertz und Ohl entwickelten Algorithmen oft eine große Menge an Konturstützpunkten erzeugt, die für die Beschreibung einer Objekthypothesenkontur nicht notwendig sind (z. B. drei Stützpunkte für eine Strecke). Wendet man den Ansatz nun auf eine Umgebung an, in der höhere Geschwindigkeiten erreicht werden, so kommt es zu Verzerrungen der Kontur, die teilweise ein Tracking der Objekthypothese nicht länger ermöglichen.

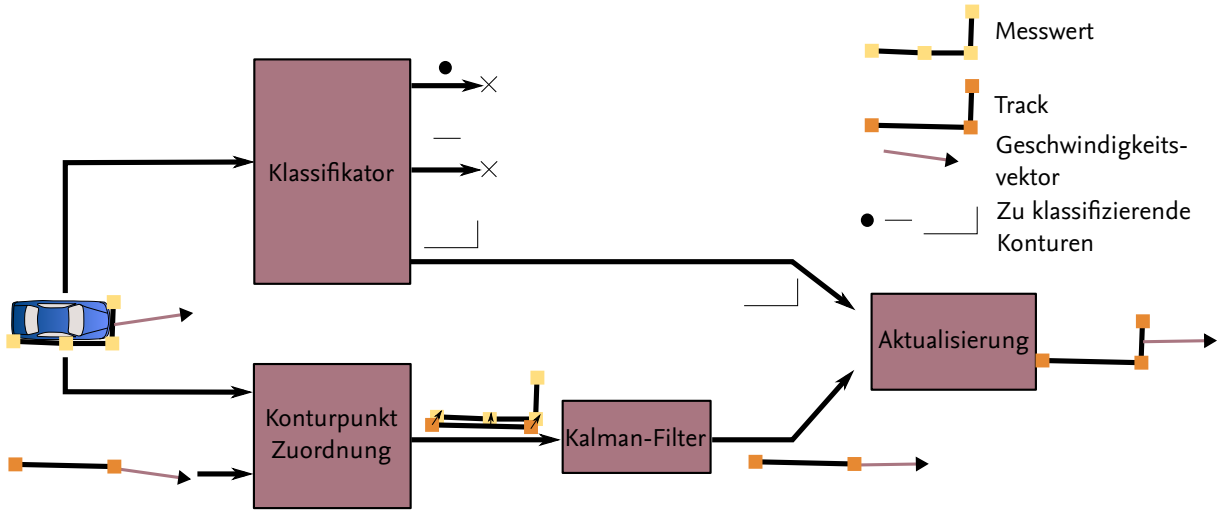


Abbildung 14.2: Datenfluss innerhalb des konturklassifizierenden Kalman-Filters

Ein direkter Einsatz des Verfahrens aus dem DARPA Urban Challenge Projekt ist somit nicht möglich. Für das hier beschriebene Filter soll, um die Anforderungen SPFA25, SPFA30 und SPFA31 erfüllen zu können, die Flexibilität des Ansatzes aus dem Team CarOLO jedoch erhalten bleiben.

Die beschriebenen Verzerrungen treten vor allem bei Objekten mit hoher Geschwindigkeit auf, da bei Objekten mit geringer Geschwindigkeit das Rauschen der Kontur gegenüber der Verschiebung durch die Bewegung überwiegt. Aus diesem Grund wurde die Filterstruktur aus Effertz (2009) durch eine Klassifikation der Objekthypothesenkontur erweitert. Abbildung 14.2 zeigt den neuen Datenfluss innerhalb des Filters. Zu sehen ist ein Messwert mit vier Konturpunkten in L-Form, der zunächst einem Klassifikator zugeführt wird. Dieser Algorithmus klassifiziert den Messwert als L-Form, welche durch drei Konturpunkte dargestellt werden kann. Parallel wird die Verschiebung des Messwerts gegenüber dem zugeordneten Track bestimmt und mit einem Kalman-Filter fusioniert. Anschließend wird das Ergebnis der Klassifikation auf den korrigierten Messwerttrack angewendet und die bestehende Kontur durch die neue vereinfachte Kontur ersetzt.

14.1.1 Entscheidung für ein Objekthypothesenmodell

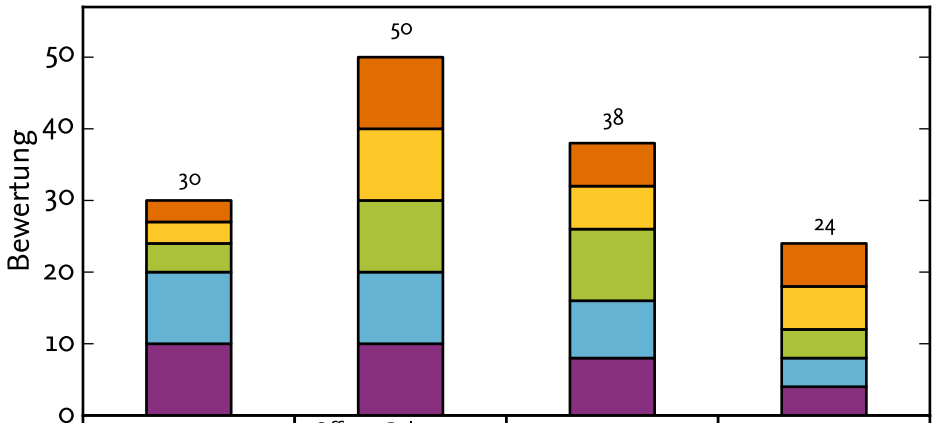
Die durch das *MainFusionModule* genutzten Sensoren liefern unterschiedliche Objekthypothesenmodelle. Sie müssen in einem Fusionsmodul auf ein gemeinsames Modell abgebildet werden. Dieses bildet das Objekthypothesenmodell des Filters des Fusionsmoduls.

Sensor	Geometriemodell	Bewegungsmodell
IBEO Alaska XT	Offener Polygonzug im 2D-Raum	Konstante Geschwindigkeit (CV-Modell)
IBEO Lux	Offener Polygonzug im 2D-Raum	Konstante Geschwindigkeit (CV-Modell)
Hella IDIS	Linker und rechter Rand der Objekthypothese (Strecke) im 2D-Raum	Keine Bewegungsinformation
SMS UMMR 2010	2D-Punkt	Konstante Geschwindigkeit (CV-Modell)
SMS UMMR 2006 (nach *RadarFusionModule*)	2D-Punkt	Konstante Geschwindigkeit (CV-Modell)

Tabelle 14.1: Durch das *MainFusionModule* genutzte Informationen der Objekthypothesenmodelle der eingesetzten Sensoren

Tabelle 14.1 zeigt die Geometrie- und Bewegungsmodelle der durch das *MainFusionModule* verarbeitenden Sensoren. Neben den durch die Sensoren genutzten Modelle gibt es noch eine Reihe weiterer gebräuchlicher Modelle. So nutzt Wender beispielsweise ein rechteckiges boxförmiges Geometriemodell (Wender, 2007) und in Blackman u. Popoli (1999, Seite 203ff) wird als Bewegungsmodell das konstante Beschleunigungsmodell sowie das Coordinated Turn Modell beschrieben.

Da offensichtlich nicht alle Sensoren das gleiche Modell nutzen, muss ein Kompromiss gefunden werden, mit dem die Eigenschaften der Sensoren ausreichend abgebildet werden können. Zur Auflösung von Konflikten in Anforderungen schlagen Pohl u. Rupp (2011, Seite 120ff) u. a. eine Entscheidungsmatrix vor. Dabei werden die Anforderungen möglichen Lösungen gegenübergestellt. Jede Lösung wird durch einen Experten in Bezug auf die Erfüllung der Anforderung mit einem Wert von 0 (nicht erfüllt) bis 10 (erfüllt) bewertet. Das Maximum der Addition aller Werte ergibt die zu verwendende Lösung.



	2D-Punkt	Offener Polygonzug	Strecke	Rechteck
IBEO Alaska XT	3	10	6	6
IBEO Lux	3	10	6	6
Hella IDIS	4	10	10	4
SMS UMMR 2010	10	10	8	4
SMS UMMR 2006	10	10	8	4

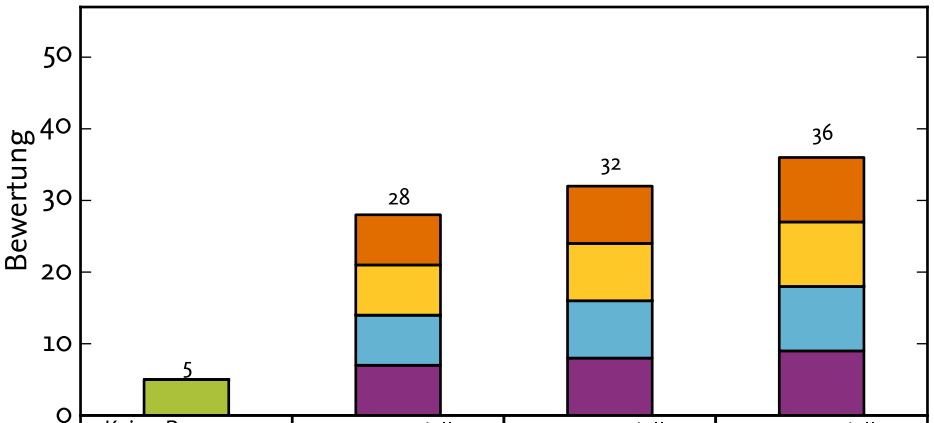
Abbildung 14.3: Entscheidungsmatrix über das eingesetzte Geometriemodell der Objekthypothesen

Als Kriterium zur Entscheidung für ein geometrisches Objekthypothesenmodell wird eine möglichst gute Abbildung der Geometriemodelle der Sensoren genutzt. Dieses bietet sich an, da einerseits alle Informationen der Sensoren genutzt werden sollen, andererseits aber keine Informationen für den Aktualisierungsschritt des Filters fehlen dürfen. Nach der Bewertung dieses Kriteriums durch den Autor ist in Abbildung 14.3 zu sehen, dass ein offener Polygonzug die Fähigkeiten der Sensoren am besten abbildet. Alle anderen Geometriemodelle schränken die Fähigkeit Objekte abzubilden ein bzw. nicht alle Sensoren liefern die benötigten Informationen zur Beschreibung eines Objekts in diesem Modell.

Zur Entscheidung für ein Objekthypothesenmodell der Objektbewegung soll neben dem Vorhandensein der nötigen Informationen zur Aktualisierung des Filters auch eine gute Repräsentation der Bewegungsmuster der Fahrzeuge im Fahrzeugumfeld genutzt werden.

Die Entscheidung für ein Bewegungsmodell ist nicht so eindeutig wie bei dem Geometriemodell. Abbildung 14.4 zeigt die Bewertung und Entscheidung für das Coordinated Turn Modell (CT) auf. Da die Fahrzeuge sich auf einer Straße, bedingt durch zwei gelenkte Räder, im Wesentlichen entlang eines Kreisbogens bewegen, bildet das CT-Modell diese Bewegung am besten ab.

Zur Beschreibung eines anderen Fahrzeugs ergibt sich das Objekthypothesenmodell damit als Kombination aus Kontur- und CT-Modell. Die Systemmatrix $F[k]$ des CT-Modells bei gegebenem Zustandsvektor $\hat{x}[k]$ mit nur einem Konturpunkt ergibt sich zu:



	Keine Bewegung	CV-Modell	CA-Modell	CT-Modell
IBEO Alaska XT	0	7	8	9
IBEO Lux	0	7	8	9
Hella IDIS	5	0	0	0
SMS UMMR 2010	0	7	8	9
SMS UMMR 2006	0	7	8	9

Abbildung 14.4: Entscheidungsmatrix über das eingesetzte Bewegungsmodell der Objekthypothesen

$$\hat{x}[k] = \begin{bmatrix} x[k] \\ y[k] \\ \alpha[k] \in [-\pi, \pi] \\ v[k] \\ \dot{v}[k] \\ \dot{\alpha}[k] \end{bmatrix} \tag{14.1}$$

$$F[k] = \begin{bmatrix} 1 & 0 & 0 & \cos\alpha[k] \cdot \Delta[k]t & \frac{1}{2} \cdot \cos\alpha[k] \cdot \Delta t[k]^2 & 0 \\ 0 & 1 & 0 & \sin\alpha[k] \cdot \Delta[k]t & \frac{1}{2} \cdot \sin\alpha[k] \cdot \Delta t[k]^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t[k] \\ 0 & 0 & 0 & 1 & \Delta t[k] & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{14.2}$$

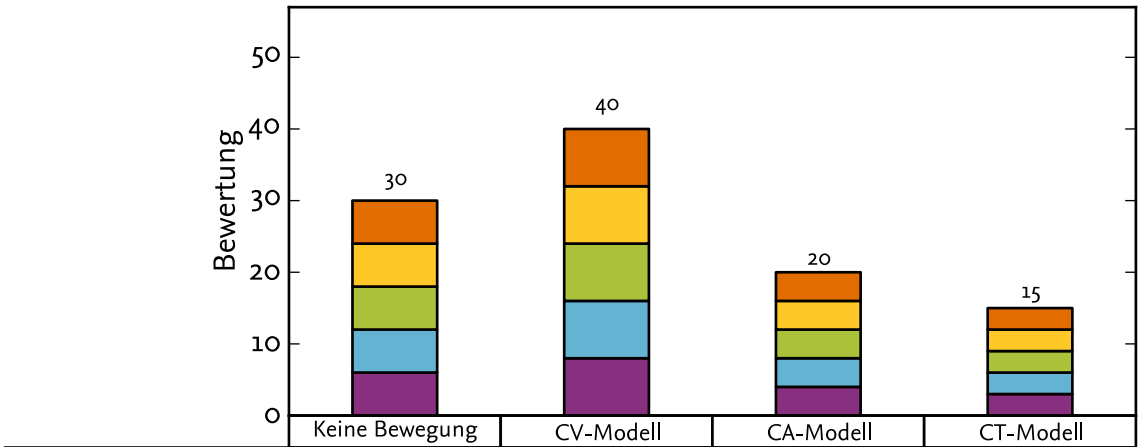
Zur Anwendung im Projekt Stadtpilot ist jedoch auch die Abbildung von Fußgängern und Randbebauung gefordert (Anforderung SPFA25 und SPFA31). Da sich Fußgänger mit sehr geringen Geschwindigkeiten bewegen, liegt der zwischen zwei Messzeitpunkten zurückgelegte Weg oft unterhalb des Rauschens eines Sensors. Bewegt sich ein Fußgänger beispielsweise mit $3\frac{km}{h}$ und der Sensor misst mit 20 Hz, so beträgt der zurückgelegte Weg nur ca. $4cm$. Da die Ortsauflösung der eingesetzten Sensoren je nach Situation nur ca. $10cm$ beträgt, ist also eine verlässliche Beobachtung der Bewegungsrichtung innerhalb eines Zeitschritts nicht mehr möglich. Darüber hinaus kann für den Fall eines statischen Objekts per Definition keine Bewegungsrichtung geschätzt werden.

Wendet man das Kriterium von Kalman zur Beobachtbarkeit (Lunze, 2010, Seite 461) auf diesen Fall an, so ist eine Beobachtbarkeit aller Zustandsgrößen auch mathematisch nicht gegeben:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{14.3}$$

$$S_B = \begin{bmatrix} HF^0 \\ \vdots \\ HF^5 \end{bmatrix} \tag{14.4}$$

$$Rang(S_B) \neq 6 \Rightarrow \text{nicht beobachtbar} \tag{14.5}$$



	Keine Bewegung	CV-Modell	CA-Modell	CT-Modell
IBEO Alaska XT	6	8	4	3
IBEO Lux	6	8	4	3
Hella IDIS	6	8	4	3
SMS UMMR 2010	6	8	4	3
SMS UMMR 2006	6	8	4	3

Abbildung 14.5: Entscheidungsmatrix über das eingesetzte Bewegungsmodell der Objekthypothesen für Fußgänger und statische Objekte

Wird die Bewertung des Bewegungsmodells unter der Berücksichtigung der Anwendbarkeit auf Fußgänger und statische Objekte vorgenommen, ergibt sich eine andere Entscheidung als in Abbildung 14.4. In Abbildung 14.5 ist zu sehen, dass sich das Konstante-Geschwindigkeitsmodell (CV) zur Abbildung von Fußgängern und statischen Objekten am besten eignet. Hierbei wird keine Bewegungsrichtung geschätzt, sondern nur eine Geschwindigkeit in X- und Y-Richtung. Für statische Objekte können diese beiden Werte die Größe 0 aufweisen. Bei Fußgängern kann hier eine Bewegung in alle Richtungen angenommen werden, was beim CT-Modell nicht möglich ist.

Die Systemmatrix ergibt sich für das CV-Modell für den Fall einer Objekthypothese mit nur einem Konturpunkt zu:

$$\hat{x}[k] = \begin{bmatrix} x[k] \\ y[k] \\ v_x[k] \\ v_y[k] \end{bmatrix} \tag{14.6}$$

$$F[k] = \begin{bmatrix} 1 & 0 & \Delta t[k] & 0 \\ 0 & 1 & 0 & \Delta t[k] \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14.7}$$

Wird nun erneut das Kalman-Kriterium zur Beobachtbarkeit angewendet, so ist zu sehen, dass eine Beobachtbarkeit aller Zustandsgrößen vorliegt:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{14.8}$$

$$S_B = \begin{bmatrix} HF^0 \\ \vdots \\ HF^3 \end{bmatrix} \tag{14.9}$$

$$Rang(S_B) = 4 \Rightarrow \text{beobachtbar} \tag{14.10}$$

Da nach den Entscheidungsmatrizen für unterschiedliche Geschwindigkeiten unterschiedliche Bewegungsmodelle sinnvoll erscheinen, wird folgendes Bewegungsmodell für das *MainFusionModule* genutzt:

Geschwindigkeit $<$ ε Konturmodell mit CV-Modell
Geschwindigkeit $\geq$ ε Konturmodell mit CT-Modell

Auf die Entscheidung über einen Wechsel zwischen den Modellen sowie deren mathematische Umsetzung wird in Abschnitt 14.1.3 näher eingegangen.

14.1.2 Konturpunktverarbeitung

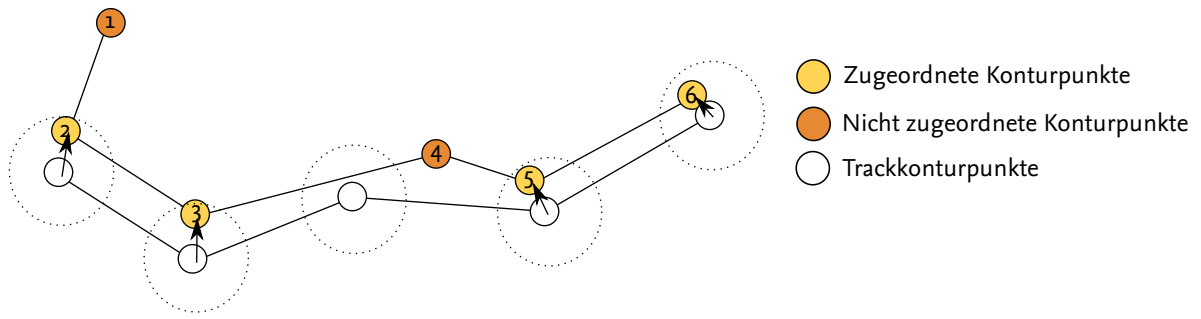


Abbildung 14.6: Konturpunktzuordnung Phase 1

```
input: ContourpointsOfTrack[], ContourpointsOfMeasurement[], ε
map = [][];
foreach pt ∈ ContourpointsOfTrack do
  foreach pm ∈ ContourpointsOfMeasurement do
    map[pt][pm] = |pt − pm|_2 // euclidean distance;
  od
od
mapping = MinimumFilter(map);
foreach m ∈ mapping do
  if (m.distance < ε) // gating
    mapPoints(m.p1, m.p2);
  fi
od
```

Programmabschnitt 14.1: Pseudocode der Phase 1 der Konturpunktzuordnung

Das durch die Fusion genutzte Kalman-Filter verarbeitet die Konturpunkte einer Objekthypothese nicht einzeln. Aus diesem Grund muss die Informationsmenge zur Verarbeitung reduziert werden. Das Filter benötigt als Größe die Verschiebung der Kontur zwischen dem auf den Messzeitpunkt prädizierten Track und der gemessenen Objekthypothese $[\Delta x, \Delta y]^T$. Im optimalen Fall hat sich die Objekthypothese entsprechend dem Bewegungsmodell bewegt und der Vektor besteht aus dem Wert null. Gründe dafür, dass dies in der Realität oft nicht der Fall ist, sind vor allem:

- Rauschen der Sensoren
- Unzulänglichkeiten in der Beschreibung des geometrischen Modells der Sensoren
- Unfähigkeit des Bewegungsmodells, die reale Bewegung zu beschreiben
- Veränderungen der messbaren Kontur des Objekts
- unterschiedliche Beschreibung der Kontur eines Objekts durch verschiedene Sensoren

Die Behandlung des Sensorrauschens sowie die Modellfehler des Objekthypothesenmodells werden durch das Kalman-Filter gewährleistet. Die Schätzung der Beschreibung beider Teile des Objekthypothesenmodells findet sich in den nächsten beiden Abschnitten. Die Herausforderung der unterschiedlichen Repräsentation einer Kontur durch verschiedene Sensoren sowie die Veränderungen der messbaren Kontur eines Objekts werden im Folgenden behandelt. Hierzu wurde das Verfahren zur Konturpunktzuordnung nach Effertz (2009, Seite 82) erweitert und zur Berechnung der Konturverschiebung angepasst. Das daraus entstandene Verfahren spaltet sich in drei Phasen auf:

1. Punkt-zu-Punkt-Zuordnung
2. Punkt-zu-Strecke-Zuordnung
3. Hinzufügen neuer Punkte

Punkt-zu-Punkt-Zuordnung

Die erste Phase der Konturpunktverarbeitung ordnet Stützpunkte des Tracks denen der Messung zu. In Abbildung 14.6 sind zwei Beispielkonturen zu sehen. Die untere Kontur stellt

```
input: ContourpointsOfTrack[], ContourpointsOfMeasurement[], ε
for i=0;i< ContourpointsOfTrack.length−1;++i do
  foreach pm ∈ NotMappedContourpointsOfMeasurement do
    pt1 = ContourpointsTrack[i];
    pt2 = ContourpointsTrack[i+1];
    if (|pm − pt1pt2|_2 < ε) // gating
      mapPoints(pt1, pm); mapPoints(pt2, pm);
    fi
  od
od
```

Programmabschnitt 14.2: Pseudocode der Phase 2 der Konturpunktzuordnung

den prädizierten Track dar. Die obere, leicht verschobene und erweiterte Kontur repräsentiert die Messung. Innerhalb des Algorithmus werden alle Punkte miteinander verglichen und mithilfe der euklidischen Abstandsfunktion ein Wert für die Ähnlichkeit bestimmt (siehe Programmabschnitt 14.1). Ist dieser Abstand, verglichen mit den Abständen zu allen anderen Konturpunkten, der niedrigste Wert und gleichzeitig unterhalb einer Schwelle ε, so wird eine Zuordnung vorgenommen. Dieser Wert findet später Eingang in den Filterschritt. Nach Anwendung des Algorithmus konnten die Konturpunkte 2, 3, 5 und 6 einem Konturpunkt der Messung zugordnet werden. Die restlichen Punkte werden im Folgenden behandelt.

Punkt-zu-Strecke-Zuordnung

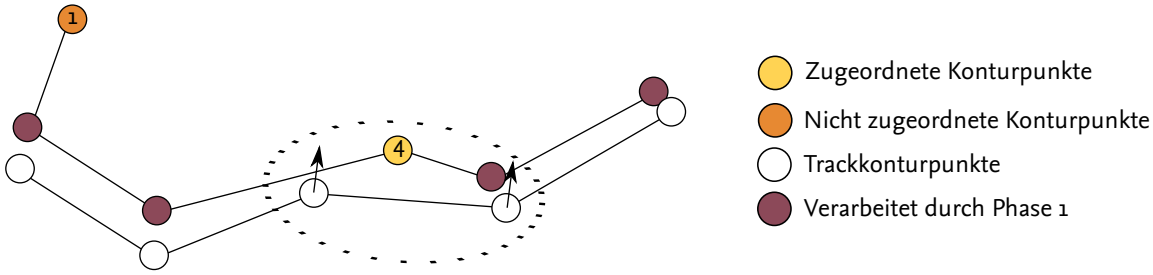


Abbildung 14.7: Konturpunktzuordnung Phase 2

Wird ein Objekt von mehr als einem Sensor wahrgenommen, ist es wahrscheinlich, dass dieser nicht exakt die gleichen Stützstellen auf der Kontur wählt wie ein anderer Sensor. Dies führt dazu, dass Punkte in Phase 1 nicht zugeordnet werden können. In Phase 2 wird deshalb eine Zuordnung dieser nicht verarbeiteten Punkte zu einem Streckenabschnitt der Kontur des Tracks vorgenommen. Abbildung 14.7 und Programmabschnitt 14.2 illustrieren den Algorithmus. Hierbei wird über alle Streckenabschnitte des Tracks iteriert und der lotrechte Abstand zwischen Strecke und Messkonturpunkt bestimmt. Ist dieser unterhalb des Schwellwerts, so wird eine Zuordnung beider Streckenendpunkte zum Messpunkt vorgenommen. In dem dargestellten Beispiel wird Konturpunkt 4 dem dritten Streckenabschnitt des Tracks zugeordnet.

Hinzufügen neuer Punkte

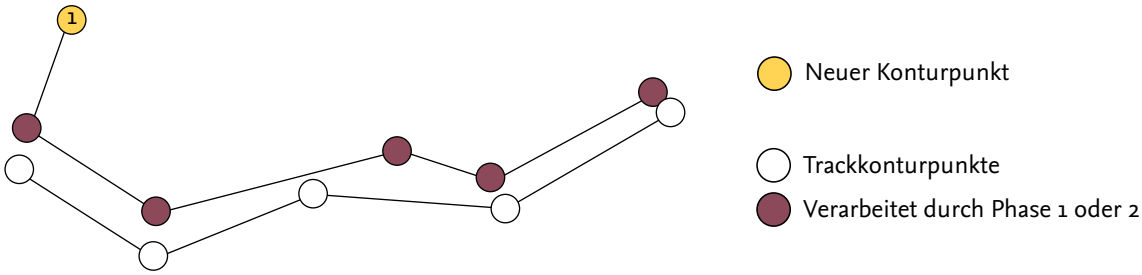


Abbildung 14.8: Konturpunktzuordnung Phase 3

Da die Messung die aktuelle Wahrnehmung des Objekts darstellt, hat sie ein höheres Gewicht als die Konturbeschreibung des Tracks. Aus diesem Grund werden Konturpunkte, die weder einem Trackkonturpunkt noch einem Konturabschnitt zugeordnet werden konnten, der Kontur des Tracks hinzugefügt. In Abbildung 14.8 wird dies mit dem Konturpunkt 1 durchgeführt. Da die Konturklassifikation der Filterstruktur die Anzahl an Konturpunkten bereits vorgibt, wird die dritte Phase nur bei Objekthypothesen durchgeführt, die nicht eindeutig klassifiziert werden konnten.

14.1.3 Schätzung des eingesetzten Bewegungsmodells

Um das in Abschnitt 14.1.1 festgelegte Bewegungsmodell umsetzen zu können, ist eine Entscheidung nötig, wann welches der beiden ausgewählten Modelle genutzt wird. Hierzu wird eine Entscheidung mithilfe eines Dempster-Shafer-Filters herbeigeführt.

Dazu wird ein Wahrscheinlichkeitspotenzraum aus den Elementen CV, CT und N (unbekannt) aufgebaut. Die Massenverteilungen M_m^t werden auf 1 normiert und dürfen aus numerischen Gründen nicht die Werte 0 oder 1 erreichen:

$$2^{\Theta} = \{N, CV, CT\} \quad (14.11)$$

$$1 = \sum_{i \in 2_m^{\Theta}} M(i) \quad (14.12)$$

$$\begin{aligned} M_m^t(N) &\in [0.05, 0.95] \\ M_m^t(CV) &\in [0.05, 0.95] \\ M_m^t(CT) &\in [0.05, 0.95] \end{aligned} \quad (14.13)$$

$\cap$	N	CV	CT
N	N	CV	CT
CV	CV	CV	$\emptyset$
CT	CT	$\emptyset$	CT

(14.14)

Für neu erstellte Objekthypothesen wird das CT-Modell angenommen und die Massenverteilung innerhalb des Wahrscheinlichkeitspotenzraums mit $M_m^t = [0.9, 0.05, 0.05]$ initialisiert.

Zur Aktualisierung der Massen ist eine weitere Massenverteilung M_m^m notwendig, die sich aus dem Track unmittelbar nach der Aktualisierung durch einen Messwert ergibt. Hierzu wird der Betrag der Differenz der Trackgeschwindigkeit $v[k]$ von einem Schwellwert ε genutzt. Bei Versuchen hat sich ein Wert von $\varepsilon = 1,5\frac{m}{s}$ als sinnvoll erwiesen.

Die Massen verteilen sich wie folgt und wurden empirisch ermittelt:

$$M_m^m(N) = 0.15 \tag{14.15}$$

$$\lambda = \begin{cases} min(|v[k]| - \varepsilon, \varepsilon) & , v[k] > 0 \\ min(|v[k]| - \varepsilon, -\varepsilon) & , v[k] \leq 0 \end{cases} \tag{14.16}$$

$$M_m^m(CV) = \begin{cases} 1 - (M_m^m(N) + M_m^m(CT)) & , \lambda > 0 \\ \frac{v[k]}{\varepsilon \cdot (1 - M_m^m(N))} & , \lambda \leq 0 \end{cases} \tag{14.17}$$

$$M_m^m(CT) = \begin{cases} \frac{v[k]}{\varepsilon \cdot (1 - M_m^m(N))} & , \lambda > 0 \\ 1 - (M_m^m(N) + M_m^m(CV)) & , \lambda \leq 0 \end{cases} \tag{14.18}$$

Die nun gewonnenen Massenverteilungen können mithilfe von Formel 2.39 zu einer neuen Massenverteilung kombiniert werden. Diese Massenverteilung bildet die Grundlage für die Entscheidung über das eingesetzte Bewegungsmodell:

$M_m^t(CV) > 0.7$ $\Rightarrow$ CV-Modell
$M_m^t(CT) > 0.7$ $\Rightarrow$ CT-Modell
sonst $\Rightarrow$ Beibehalten des aktuellen Bewegungsmodells

Zum Übergang von einem Bewegungsmodell in ein anderes müssen der Zustandsvektor $\hat{x}[k]$ und die Kovarianzmatrix $P[k]$ in dieses überführt werden. Die Konvertierung des Zustandsvektors ergibt sich durch eine Transformation von der polaren in die kartesische Darstellung bzw. umgekehrt (Bronstein u. a., 2001, Seite 195ff):

$$x_{\hat{CV}}[k] = \begin{bmatrix} v_x[k] = \cos(\alpha[k]) \cdot v[k] \\ v_y[k] = \sin(\alpha[k]) \cdot v[k] \end{bmatrix} \tag{14.19}$$

$$x_{\hat{CT}}[k] = \begin{bmatrix} v[k] = \sqrt{v_x[k]^2 + v_y[k]^2} \\ \alpha[k] = \begin{cases} \arctan\frac{v_y[k]}{v_x[k]} + \pi & , v_x[k] < 0 \\ \arctan\frac{v_x[k]}{v_y[k]} & , v_x[k] > 0 \\ \frac{\pi}{2} & , v_x[k] = 0 \wedge v_y[k] > 0 \\ -\frac{\pi}{2} & , v_x[k] = 0 \wedge v_y[k] < 0 \\ \text{undefiniert} & , v_x[k] = v_y[k] = 0 \end{cases} \\ \dot{v}[k] = 0 \\ \dot{\alpha}[k] = 0 \end{bmatrix} \tag{14.20}$$

Die Transformation der Kovarianzmatrix $P[k]$ von kartesischen in Polarkoordinaten lässt sich über einen Tensor herleiten und ergibt sich damit zu:

$$F_{CV \to CT}[k] = \begin{bmatrix} \frac{v_x[k]}{v_x[k]^2 + v_y[k]^2} & -\frac{v_y[k]}{v_x[k]^2 + v_y[k]^2} \\ \frac{v_y[k]}{\sqrt{v_x[k]^2 + v_y[k]^2}} & \frac{v_x[k]}{\sqrt{v_x[k]^2 + v_y[k]^2}} \end{bmatrix} \tag{14.21}$$

$$P_{CT}[k] = F_{CV \to CT}[k] \cdot P_{CV}[k] \cdot F_{CV \to CT}[k]^T \tag{14.22}$$

Die Transformation der Kovarianzmatrix von Polarkoordinaten in kartesische wurde von Lerro u. Bar-Shalom ausführlich hergeleitet und ergibt sich zu (Lerro u. Bar-Shalom, 1993):

$$F_{CT\rightarrow CV}[k] = \begin{bmatrix} \cos(\alpha[k]) \cdot v[k] & \sin(\alpha[k]) \\ -\sin(\alpha[k]) \cdot v[k] & \cos(\alpha[k]) \end{bmatrix} \tag{14.23}$$

$$P_{CV}[k] = F_{CT\rightarrow CV}[k] \cdot P_{CT}[k] \cdot F_{CV\rightarrow CV}[k]^T \tag{14.24}$$

Das für die Beschreibung der Bewegungsrichtung des CV-Modells eingesetzte Polarkoordinatensystem repräsentiert den Winkel $0°$ durch den Wert $n \cdot 2\pi, n \in \mathbb{Z}$. Bewegt sich die Objekthypothese nun ungefähr in Richtung $0°$, so schwankt der Wert für $\alpha[k]$ bedingt durch das Rauschen der Objekthypothese zwischen 0 und 2π. Da diese Werte gleichbedeutend sind, kommt es in der Praxis zu großen Winkelgeschwindigkeiten mit einem Vielfachen von 2π. Aus diesem Grund muss für das Kalman-Filter eine Transformation des Koordinatensystems durchgeführt werden. Hierzu wird das Koordinatensystem intern um π gedreht, sobald sich die Bewegungsrichtung der Objekthypothese in der rechten Hälfte der Polarebene befindet.

14.1.4 Schätzung des eingesetzten Geometriemodells

Der in Abschnitt 14.1 vorgestellte Datenfluss innerhalb des konturklassifizierenden Kalman-Filters bestimmt den Typ einer Kontur und schränkt anschließend das eingesetzte geometrische Konturmodell ein. Dies resultiert in einer Reduktion von Konturverzerrungen und somit einer erhöhten Güte der Abbildungseigenschaften der Objekthypothesen.

Zur Klassifikation wurden Konturen ausgewählt, die im Testfeld des Stadtpiloten häufig vorkommen. Diese zu klassifizierenden Konturen werden im Folgenden Meta-Konturen genannt.

Die Abbildungen 14.9 und 14.10 zeigen die Verteilung der Häufigkeit der Konturen Punkt, Strecke, L-Kontur und U-Kontur für die Sensoren IBEO Alaska XT, Hella IDIS und SMS UMMR 2010 bei einer Fahrt auf dem Braunschweiger Stadtring. Die eine Darstellung beschreibt die Verteilung im Bereich bis ca. $20m$ vor dem Fahrzeug, die andere Darstellung beschreibt die Verteilung ab ca. $60m$ vor dem Fahrzeug. Da der Sensor SMS UMMR 2010 Objekthypothesen mit Punkt-Modell erzeugt, werden auch alle Objekthypothesen entsprechend klassifiziert. Der Hella IDIS-Sensor repräsentiert Objekthypothesen durch Strecken. Hierbei ergibt sich ein Überhang zu Strecken-Meta-Konturen. Die Punkt-Meta-Konturen des Sensors ergeben sich aus der Klassifikation von sehr kurzen Strecken als Punkt-Objekthypothesen. Der IBEO Alaska XT Sensor produziert Objekthypothesen, welche Objekte durch Polygone beschreiben. Aus diesem Grund ist die Verteilung auf die unterschiedlichen Meta-Konturen deutlich differenzierter. Zwischen Punkt- und Strecken-Konturen ergibt sich fast eine Gleichverteilung während Objekthypothesen mit L-Kontur weniger häufig vorkommen. Hierbei spielt auch die Entfernung vom Sensor eine erhebliche Rolle. So treten in der Verteilung der Meta-Konturen in einer Entfernung ab $60m$ (Abbildung 14.10) keine L-Konturen mehr auf. Dies lässt sich auf das Messprinzip und die Objekthypothesenbildung des Sensors zurückführen. So ist, bedingt durch die Winkelauflösung des Sensors, der Abstand zwischen zwei Messpunkten hier erheblich größer und damit nicht mehr für eine genaue Konturbeschreibung ausreichend.

Um die Konturen nach den Meta-Konturen zu klassifizieren, wurden diese mithilfe des von Arkin u. a. beschriebenen Algorithmus untersucht (Arkin u. a., 1991). Dabei kommt die Turning-Funktion $\Theta_A(s)$ zum Einsatz. Diese beschreibt den Polygonzug der Kontur A

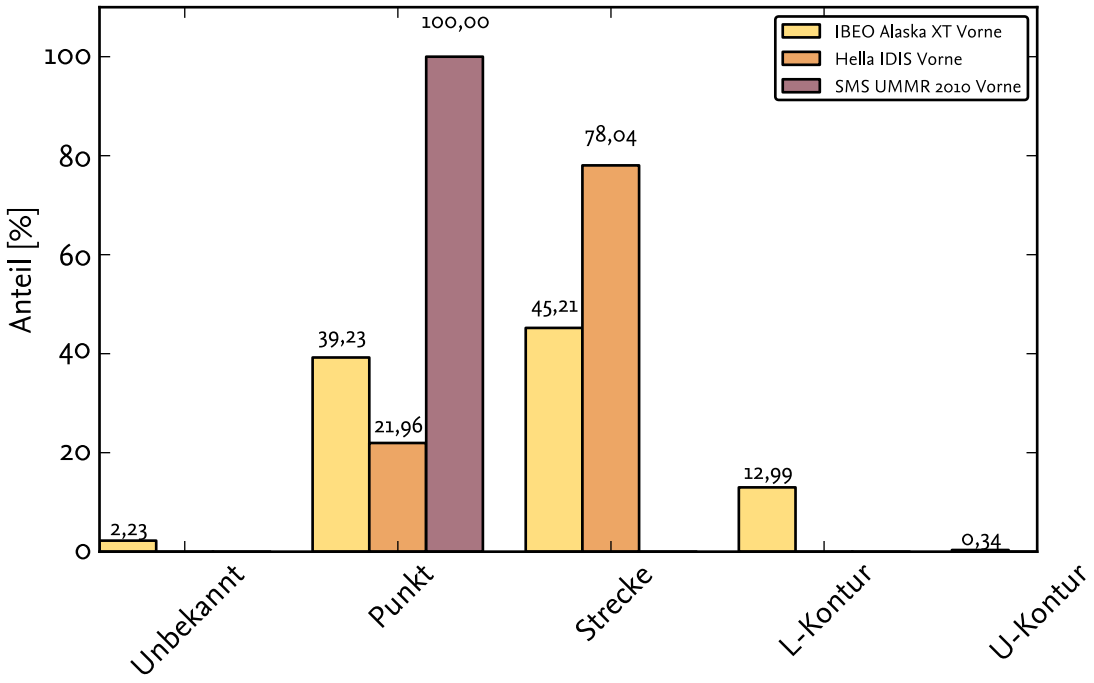


Abbildung 14.9: Übersicht über die Häufigkeit von ausgewählten Meta-Konturen bei unterschiedlichen Sensoren bis $20m$ vor dem Fahrzeug während einer Messfahrt auf dem Braunschweiger Stadtring

durch den Winkel zwischen zwei Schenkeln im Uhrzeigersinn. Die Länge eines Schenkels wird dabei auf die Länge 1 normiert. Abbildung 14.11 zeigt die Turning-Funktion für eine U-Kontur. Diese weist drei Schenkel auf, zwischen denen je 90° Winkeländerung liegen. Wird die Turning-Funktion für zwei Konturen aufgestellt, so lässt sich mithilfe eines Integrals ein Ähnlichkeitswert bestimmen. Die so gewonnenen Werte können genutzt werden, um mit der Evidenztheorie eine stabile Schätzung der Objekthypothesenkontur vorzunehmen.

$$D_B^A = \int_0^1 |\Theta_A(s) - \Theta_B(s)| ds \tag{14.25}$$

Zur Beschreibung eines Dempster-Shafer-Filters ist, wie bei der Schätzung des Bewegungsmodells, ein Wahrscheinlichkeitspotenzraum notwendig. Dieser enthält alle Meta-Konturen aus Abbildung 14.9 sowie die unbekannte Kontur N. Da L-Konturen durch die Turning-Funktion auf vier verschiedene Arten dargestellt werden können, finden sich diese Typen auch im Wahrscheinlichkeitspotenzraum wieder.

$$2^{\Theta}_{Polyline} = \{N, \bullet, |, \rfloor, \llcorner, \lfloor, \lrcorner, \sqcup\} \tag{14.26}$$

$$1 = \sum_{i \in 2^{\Theta}_{Polyline}} M(i) \tag{14.27}$$

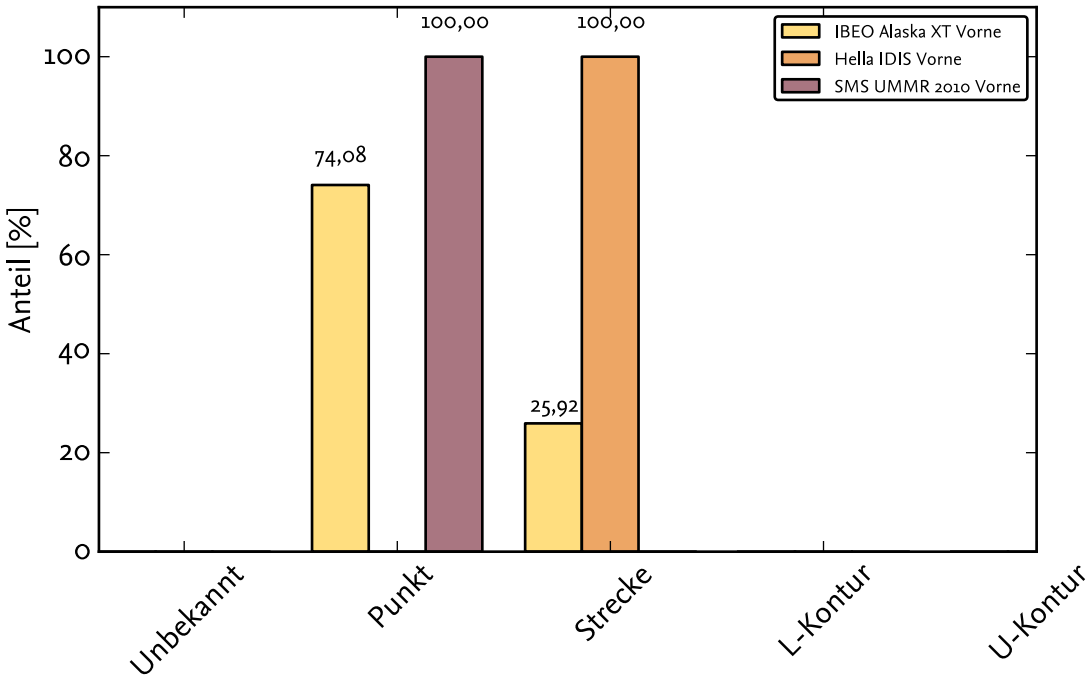


Abbildung 14.10: Übersicht über die Häufigkeit von ausgewählten Meta-Konturen bei unterschiedlichen Sensoren ab $60m$ vor dem Fahrzeug während einer Messfahrt auf dem Braunschweiger Stadtring

$$\begin{array}{lcllcl} M(N) & \in & [0.1, 0.95] & M(\rfloor) & \in & [0.00, 0.95] \\ M(\bullet) & \in & [0.00, 0.95] & M(\lfloor) & \in & [0.00, 0.95] \\ M(|) & \in & [0.00, 0.95] & M(\sqcup) & \in & [0.00, 0.95] \\ M(\llcorner) & \in & [0.00, 0.95] & M(\lrcorner) & \in & [0.00, 0.95] \end{array} \tag{14.28}$$

$\cap$	N	$\bullet$	$\|$	$\rfloor$	$\llcorner$	$\lfloor$	$\lrcorner$	$\sqcup$
N	N	$\bullet$	$\|$	$\rfloor$	$\llcorner$	$\lfloor$	$\lrcorner$	$\sqcup$
$\bullet$	$\bullet$	$\bullet$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\|$	$\|$	$\emptyset$	$\|$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\rfloor$	$\rfloor$	$\emptyset$	$\emptyset$	$\rfloor$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\llcorner$	$\llcorner$	$\emptyset$	$\emptyset$	$\emptyset$	$\llcorner$	$\emptyset$	$\emptyset$	$\emptyset$
$\lfloor$	$\lfloor$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\lfloor$	$\emptyset$	$\emptyset$
$\lrcorner$	$\lrcorner$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\lrcorner$	$\emptyset$
$\sqcup$	$\sqcup$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\sqcup$

(14.29)

Die für die Kombinationsregel von Dempster (Formel 2.39) notwendige zweite Massenverteilung ergibt sich aus der Ähnlichkeit zwischen der gemessenen Kontur A und den Meta-Konturen D_B^A. Dabei wird eine maximale Abweichung zwischen Messwert und Meta-Kontur von $\varepsilon = 1$ zugelassen. Die Massenverteilung wird anschließend normiert.

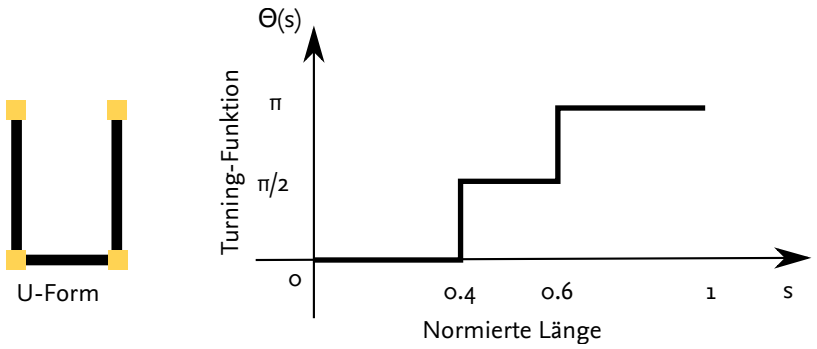


Abbildung 14.11: Turningfunktion einer U-Kontur

$$\begin{array}{llll} M(N) = 0.1 & & M(\bullet) = \begin{cases} 0.3 & \text{, Länge} < 0.3m \\ 0 & \text{, sonst} \end{cases} \\ M(|) = D^A_{|} & & M(\rfloor) = D^A_{\rfloor} \\ M(\llcorner) = D^A_{\llcorner} & & M(\lfloor) = D^A_{\lfloor} \\ M(\lrcorner) = D^A_{\lrcorner} & & M(\sqcup) = D^A_{\sqcup} \end{array} \tag{14.30}$$

Da nicht alle Objekthypothesenmodelle alle Meta-Konturen abbilden können, kann dieses Verfahren nicht für alle Sensoren genutzt werden. So kann beispielsweise eine Objekthypothese eines Sensors mit Strecken-Modell sowohl ein Objekt mit Strecken-Kontur als auch ein Objekt mit einer L-Kontur beschreiben. Aus diesem Grund müssen die Wahrscheinlichkeitspotenzräume für Sensoren mit Punkt- und Strecken-Modellen angepasst werden. Für diese Sensoren repräsentiert das Element mit den detailliertesten Abbildungseigenschaften nun die Elemente des Wahrscheinlichkeitspotenzraums $2^{\Theta}_{Polyline}$, die nicht durch den Sensor abgebildet werden können.

Für den Fall eines Sensors mit Punkt-Modell ergibt sich folgender Wahrscheinlichkeitspotenzraum:

$$2^{\Theta}_{Punkt} = \{N, \{\bullet, |, \rfloor, \llcorner, \lfloor, \lrcorner, \sqcup\}\} \tag{14.31}$$

$$1 = \sum_{i \in 2^{\Theta}_{Punkt}} M(i) \tag{14.32}$$

$\cap$	N	$\bullet$	$\|$	$\rfloor$	$\llcorner$	$\lfloor$	$\lrcorner$	$\sqcup$
N	N	$\bullet$	$\|$	$\rfloor$	$\llcorner$	$\lfloor$	$\lrcorner$	$\sqcup$
$\{\bullet, \|, \rfloor, \llcorner, \lfloor, \lrcorner, \sqcup\}$	$\{\bullet, \|, \rfloor, \llcorner, \lfloor, \lrcorner, \sqcup\}$	$\bullet$	$\|$	$\rfloor$	$\llcorner$	$\lfloor$	$\lrcorner$	$\sqcup$

(14.33)

Die Aufstellung der Massenverteilung auf dem Messdatum eines Sensors ist wie folgt definiert:

$$M(N) = 0.1 \qquad M(\{\bullet, |, \rfloor, \llcorner, \lfloor, \lrcorner, \sqcup\}) = 0.9 \tag{14.34}$$

Für den Fall eines Sensors mit Strecken-Modell ergibt sich folgender Wahrscheinlichkeitspotenzraum:

$$2^{\Theta}_{Strecke} = \{N, \bullet, \{|, \rfloor, \llcorner, \lfloor, \lrcorner, \sqcup\}\} \tag{14.35}$$

$$1 = \sum_{i \in 2^{\Theta}_{Strecke}} M(i) \tag{14.36}$$

$\cap$	N	$\bullet$	$\vert$	$\rfloor$	$\llcorner$	$\lfloor$	$\lrcorner$	$\sqcup$
N	N	$\bullet$	$\vert$	$\rfloor$	$\llcorner$	$\lfloor$	$\lrcorner$	$\sqcup$
$\bullet$	$\bullet$	$\bullet$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\{\vert,\rfloor,\llcorner,\lfloor,\lrcorner,\sqcup\}$	$\{\vert,\rfloor,\llcorner,\lfloor,\lrcorner,\sqcup\}$	$\emptyset$	$\vert$	$\rfloor$	$\llcorner$	$\lfloor$	$\lrcorner$	$\sqcup$

(14.37)

Die Aufstellung der Massenverteilung für das Messdatum eines Sensors ist wie folgt definiert:

$$M(N) = 0.1 \qquad M(\bullet) = \begin{cases} 0.8 & ,\text{Länge} < 0,3m \\ 0 & ,\text{sonst} \end{cases}$$
$$M(\vert) = \begin{cases} 0.1 & ,\text{Länge} < 0,3m \\ 0.9 & ,\text{sonst} \end{cases} \tag{14.38}$$

Eine Entscheidung für eine Meta-Kontur erfolgt über die Support-Funktion aus Formel 2.41. Hierzu wird das Maximum über alle Meta-Konturen bestimmt.

14.1.5 Erweitertes Kalman-Filter

Da die in Abschnitt 14.1.3 festgelegten Bewegungsmodelle nicht vollständig linear sind, wird zur Aktualisierung des Tracks ein erweitertes Kalman-Filter mit Mode-Switch (Li u. Bar-Shalom, 1992) im Bewegungsmodell eingesetzt. Je nach eingesetztem Bewegungsmodell wird hierzu ein unterschiedlicher Zustandsvektor genutzt:

$$x_{\hat{CT}}[k] = \begin{bmatrix} \Delta x[k] = 0 \\ \Delta y[k] = 0 \\ \alpha[k] \in [-\pi, \pi] \\ v[k] \\ \dot{v}[k] \\ \dot{\alpha}[k] \end{bmatrix} \tag{14.39}$$

$$x_{\hat{CV}}[k] = \begin{bmatrix} \Delta x[k] = 0 \\ \Delta y[k] = 0 \\ v_x[k] \\ v_y[k] \end{bmatrix} \tag{14.40}$$

Die Zustandsbeschreibung des Tracks im Filter ist stark vereinfacht und bildet nicht die vollständige Kontur ab. Effertz hat gezeigt, dass durch die Reduktion des Zustands auf den Schwerpunkt der Kontur (Centroid) Probleme aus Mehrdeutigkeiten, die bei der Stützstellenzuordnung aus Abschnitt 14.1.2 auftreten, aufgelöst werden können (Effertz, 2009, Seite 89ff). Die dabei entstehenden Abbildungsfehler haben in der Praxis keine Bedeutung. Dadurch, dass die Konturpunktzuordnung die Abweichung der Kontur des Tracks von der des Messwerts bestimmt, wird durch das Filter ebenfalls diese Verschiebung geschätzt. Sie wird zu Beginn eines Filterschritts als 0 angenommen, da der Track seit dem letzten Zeitpunkt t keine Verschiebung erfahren hat.

Basierend auf dem Bewegungsmodell ergeben sich die Systemübergangsfunktionen f_{CT} und f_{CV} zu:

$$\begin{array}{rcl}
\Delta x[k+1] & = & \begin{cases} \Delta x[k] + v[k] \cdot \cos(\alpha[k]) \cdot \Delta t + \frac{1}{2} \cdot a[k] \cdot \cos(\alpha[k]) \cdot \Delta t^2 & \text{, CT-Modell} \\ \Delta x[k] + v_x[k] \cdot \Delta t & \text{, CV-Modell} \end{cases} \\
\Delta y[k+1] & = & \begin{cases} \Delta y[k] + v[k] \cdot \sin(\alpha[k]) \cdot \Delta t + \frac{1}{2} \cdot a[k] \cdot \sin(\alpha[k]) \cdot \Delta t^2 & \text{, CT-Modell} \\ \Delta y[k] + v_y[k] \cdot \Delta t & \text{, CV-Modell} \end{cases} \\
\alpha[k+1] & = & \alpha[k] \\
v[k+1] & = & v[k] + a[k] \cdot \Delta t \\
a[k+1] & = & a[k] \\
v_x[k+1] & = & v_x[k] \\
v_y[k+1] & = & v_y[k]
\end{array} \tag{14.41}$$

Zur Anwendung im erweiterten Kalman-Filter ist die Jakobi-Matrix $F[k]$ für beide Bewegungsmodelle zu bestimmen:

$$\begin{array}{rcl}
F^{CV}[k] & = & \frac{\partial f_{CV}(\hat{x}[k])}{\partial(\Delta x[k], \Delta y[k], v_x[k], v_y[k])} = \begin{bmatrix} 1 & 0 & \Delta t[k] & 0 \\ 0 & 1 & 0 & \Delta t[k] \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
F^{CT}[k] & = & \frac{\partial f_{CT}(\hat{x}[k])}{\partial(\Delta x[k], \Delta y[k], \alpha[k], v[k], \dot{v}[k], \dot{\alpha}[k])} \\
& = & \begin{bmatrix} 1 & 0 & 0 & \cos(\alpha[k]) \cdot \Delta t[k] & \frac{1}{2}\cos(\alpha[k]) \cdot \Delta t[k]^2 & 0 \\ 0 & 1 & 0 & \sin(\alpha[k]) \cdot \Delta t[k] & \frac{1}{2}\sin(\alpha[k]) \cdot \Delta t[k]^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t[k] \\ 0 & 0 & 0 & 1 & \Delta t[k] & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}
\end{array} \tag{14.42}$$

Die Abbildung des Messvektors $y[k]$ auf den Zustandsvektor muss ebenfalls für beide Bewegungsmodelle angegeben werden. Je nach Messgrößen des Sensors steht eine Information über den Bewegungszustand der Objekthypothesen zur Verfügung oder nicht. Diese Verfügbarkeit muss in der Abbildung berücksichtigt werden:

$$\begin{array}{rcll}
H[k]^{CV} & = & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} & \text{, bei } y[k] = [\Delta x[k], \Delta y[k]]^T \\
H[k]^{CV} & = & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \text{, bei } y[k] = [\Delta x[k], \Delta y[k], v_x[k], v_y[k]]^T \\
H[k]^{CT} & = & \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} & \text{, bei } y[k] = [\Delta x[k], \Delta y[k]]^T \\
H[k]^{CT} & = & \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} & \text{, bei } y[k] = [\Delta x[k], \Delta y[k], \alpha[k], v[k]]^T
\end{array} \tag{14.43}$$

Nach der Anwendung des Schätzalgorithmus aus Abschnitt 2.3.1 wird die Aktualisierung der Objekthypothesenkontur durchgeführt. Hierbei wird zwischen drei Fällen unterschieden:

1. Der Stützpunkt der Kontur wurde durch die Phasen 1 oder 2 der Konturpunktverarbeitung zugeordnet.
2. Der Stützpunkt konnte nicht zugeordnet werden.
3. Der Stützpunkt wurde in Phase 3 der Konturpunktverarbeitung der Kontur neu hinzugefügt.

Im Falle, dass ein Konturpunkt einem Messpunkt zugeordnet werden konnte, wird die Kalman Verstärkung $K[k]$ der individuellen Verschiebung des Punktes $(\Delta x_i, \Delta y_i)$ zu seinem korrespondierenden Punkt genutzt, um die neue Lage des Punktes (x_i, y_i) zu berechnen:

$$x_i[k|k] = x_i[k|k-1] + K[k] \cdot \Delta x_i[k] \quad (14.44)$$
$$y_i[k|k] = y_i[k|k-1] + K[k] \cdot \Delta y_i[k] \quad (14.45)$$

Konnte ein Konturpunkt keinem Punkt des Messwerts zugeordnet werden, so wird der durch das Kalman-Filter korrigierte Zustandsvektor genutzt:

$$x_i[k|k] = x_i[k|k-1] + K[k] \cdot \hat{x}[k|k]\{1\} \quad (14.46)$$
$$y_i[k|k] = y_i[k|k-1] + K[k] \cdot \hat{x}[k|k]\{2\} \quad (14.47)$$

Neu erzeugte Konturpunkte werden nicht durch das Kalman-Filter behandelt, da sie einen aktuellen Messwert repräsentieren.

14.1.6 Anpassung der Objekthypothesenkontur

Der letzte Schritt innerhalb des Ablaufs des konturklassifizierenden Kalman-Filters ist die Anpassung des Geometriemodells der Objekthypothese aufgrund einer Änderung der Schätzung mit dem Dempster-Shafer-Filter aus Abschnitt 14.1.4. Hierzu wird die Objekthypothesenkontur durch die klassifizierte Kontur ersetzt, sobald die Meta-Kontur-Schätzung zu einem anderen Ergebnis kommt als im vorhergehenden Filterschritt. Dabei muss die klassifizierte Meta-Kontur entsprechend der Daten des aktuellen Messwerts nach folgenden Kriterien angepasst werden:

- Lage
- Rotation
- Skalierung

Konturanpassung mithilfe der Turningfunktion

Die Anpassung der Rotation kann mithilfe der Turningfunktion durchgeführt werden. Dabei wird die Rotationsinvarianz der Funktion durch einen virtuellen Schenkel vom ersten Stützpunkt zum Ursprung des Koordinatensystems aufgehoben. Die Rotation des Messwerts wirkt sich anschließend in einer Verschiebung auf der Y-Achse aus (siehe Abbildung 14.12). Durch Berechnung der Funktion 14.25 kann nun für verschiedene Rotationen der Meta-Kontur eine Ähnlichkeit der Rotationen bestimmt werden (z. B. durch binäre Suche). Da

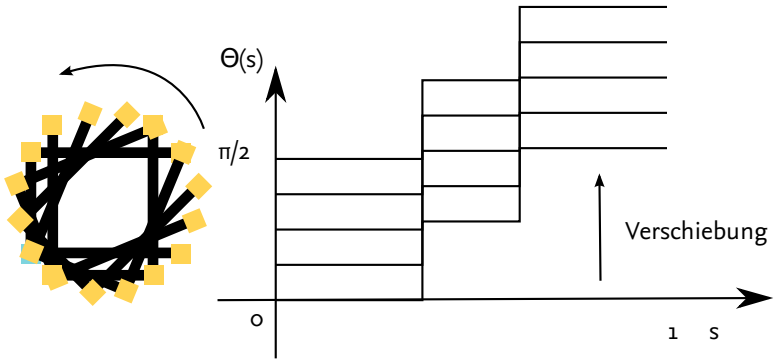


Abbildung 14.12: Bestimmung der Rotation durch Minimierung der Ähnlichkeitsfunktion $D_B^{A'}$. Rotiere Kontur A', bis sie mit Kontur B übereinstimmt.

die Messkontur nicht exakt der Meta-Kontur entspricht, ist das Minimum der Abweichung nicht zwangsläufig die Rotation des ersten Schenkels der Messkontur. Die Skalierung der Meta-Kontur kann über die Multiplikation mit der Länge der Messkontur geschehen. Zur Anpassung der Lage wird die skalierte und rotierte Kontur an die Stelle der Messung verschoben.

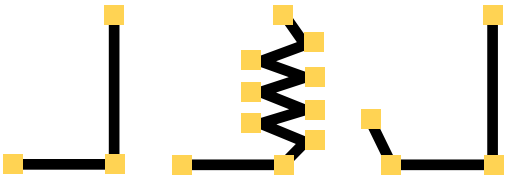


Abbildung 14.13: L-Kontur ohne (links) und mit (Mitte) hochfrequentem Rauschen, L-Kontur mit kurzer Erweiterung (rechts)

Wird die Konturanpassung auf diese Weise durchgeführt, so ergeben sich, bedingt durch Rauschen der Kontur, zwei Schwierigkeiten (siehe Abbildung 14.13):

Beeinflussung der Konturlänge durch hochfrequentes Rauschen: Durch hochfrequentes Rauschen der Messkontur verlängert sich die Gesamtlänge erheblich. Die Ähnlichkeit der Kontur zur Meta-Kontur verändert sich dahingegen kaum. Durch den Einsatz des beschriebenen Verfahrens wird in diesem Fall eine Kontur erzeugt, die deutlich länger ist und somit nicht den Messwert repräsentiert.

Kurze Erweiterungen am Anfang führen zu falscher Rotation: Sieht ein Sensor die hintere Kante eines nahen Fahrzeugs, so bildet die Objekthypothesenbildung der Sensoren IBEO Alaska XT und IBEO Lux zum Teil die Rundung des Hecks von modernen Fahrzeugen mit ab. Dies äußert sich in einer kurzen Ergänzung zur L- oder Strecken-Kontur. Bei der Bestimmung des optimalen Rotationswinkels der Meta-Kontur kann nun der falsche Winkel bestimmt werden. Letztlich wird eine stark rotierte Meta-Kontur erzeugt, die nicht den Messwert repräsentiert.

```
input: MeasurementContour[]
Output: Contur[]
bestHypothesis = 0;
for i = 0;i<MeasurementContour.length;i++ do
  for j = i;j<MeasurementContour.length;j++ do
    thisHypothesis = 1;
    thisHypothesis *= CalcMetricA(i,j);
    ...
    thisHypothesis *= CalcMetricN(i,j);
    if thisHypothesis > bestHypothesis
      bestHypothesis = thisHypothesis;
      Contour = [i,j];
    fi
  od
od
```

Programmabschnitt 14.3: Pseudocode des Konturanpassungsalgorithmus für eine Strecken-Kontur

Konturanpassung auf Basis der Stützpunkte der Messkontur

Um diese Probleme zu vermeiden, wurde ein Verfahren auf der Basis der Stützpunkte der Messkontur entwickelt. Dem Algorithmus liegt der Gedanke zugrunde, dass die Stützpunkte der Messkontur eine hohe Relevanz für die Beschreibung der Kontur besitzen. Sie sollten die Grundlage zur Beschreibung der späteren Objekthypothesenkontur des Tracks bilden.

Zur Bestimmung der notwendigen Stützpunkte der Messkontur wird der Algorithmus aus Programmabschnitt 14.3 angewandt. Er beschreibt den Fall der Erzeugung einer Strecken-Kontur. Hierbei wird über alle Stützpunkte der Messkontur iteriert, für jede Kombination an Stützpunkten eine Reihe von Metriken berechnet und der Wert mit der größten Übereinstimmung bildet die gesuchte Kontur. Durch die Nutzung der Messkonturstützpunkte wird die Anpassung der Rotation und der Skalierung durch den Algorithmus in einem Schritt durchgeführt. Die Anpassung der Lage geschieht durch eine Verschiebung der erzeugten Kontur auf die Position der Messkontur.

Je nach Meta-Kontur werden unterschiedliche Metriken genutzt, um die Eigenschaften der Kontur wiedergeben zu können. Jede bildet ihre Eigenschaften auf einen Wert im Intervall $(0,1]$ ab. Folgende Metriken habe sich in Versuchen als sinnvoll erwiesen:

Länge: Das Ziel der Metrik ist, möglichst viele Punkte der Messkontur durch die Meta-Kontur zu beschreiben. Da die Stützpunkte in der Reihenfolge der Messkontur verbleiben, soll die Metrik ihren optimalen Wert erreichen, wenn der Start- und Endstützpunkt für die Trackkontur genutzt wird.

Distanz: Um die Güte der Näherung einer durch zwei Stützpunkte beschriebenen Strecke zur Messkontur zu bestimmen, wird der lotrechte Abstand aller Punkte berechnet, welche sich zwischen den Stützpunkten befinden. Die Abstände links und rechts der Geraden werden sowohl vorzeichenbehaftet als auch dem Betrag nach aufaddiert und schließlich der Mittelwert beider Werte als Metrik genutzt.

Winkel zwischen zwei Schenkeln: Der Winkel zwischen zwei Schenkeln, die durch die Auswahl von Messkonturpunkten entstehen, muss dem der Meta-Kontur entsprechen. Hierzu wird der reale Winkel mit dem durch die Meta-Kontur vorgegebenen verglichen. In Abhängigkeit einer maximal zulässigen Abweichung wird der Wert der Metrik bestimmt.

Proportion: Bei der Anpassung der Meta-Kontur sollten die Proportionen zweier Schenkel erhalten bleiben. Dabei wird die Länge beider Schenkel bestimmt. Der Unterschied der Quotienten der Mess- und Meta-Kontur bildet sich auf den Wert der Metrik ab.

Der Algorithmus aus Programmabschnitt 14.3 kann nur für eine Strecken-Kontur genutzt werden. Durch Rekursion ist jedoch ein Ablauf für jede Meta-Kontur möglich. Für die hier verwendeten Meta-Konturen werden folgende Metriken mit entsprechender Rekursionstiefe eingesetzt (siehe Tabelle 14.2).

	Länge	Distanz	Winkel	Proportion	Rekursionstiefe
Punkt-Kontur	✗	✗	✗	✗	0
Strecken-Kontur	✓	✓	✗	✗	0
L-Kontur	✓	✓	✓	✓	1
U-Kontur	✓	✓	✓	✓	2

Tabelle 14.2: Eingesetzte Metriken und genutzte Rekursionstiefe zur Anpassung der Objekthypothesenkontur bei unterschiedlichen Meta-Konturen

14.2 Tracking zur Stabilitätsanalyse von Objekthypothesen

Unter der Stabilität einer Objekthypothese wird im Folgenden die Eigenschaft verstanden, eine Objekthypothese für den Aufenthalt im Erfassungsbereich der Sensoren kontinuierlich entsprechend eines Modells durch einen Zustandsvektor beschreiben zu können. Für das Tracking mit dem konturklassifizierenden Kalman-Filter stellt dies eine Voraussetzung zur erfolgreichen Verarbeitung dar. Treten Objekthypothesen nur für einen sehr kurzen Zeitraum auf oder verändern sich die Elemente des Zustandsvektors und der Kontur nicht entsprechend dem Bewegungs- und Geometriemodell, so ist eine erfolgreiche Zuordnung zur späteren Fusion im *MainFusionModule* nicht möglich.

Aus diesem Grund wird ein weiteres Fusionsmodul auf Basis der Softwarearchitektur realisiert, das drei Ziele verfolgt:

- Objekthypothesen müssen ein gewisses Mindestmaß an Stabilität aufweisen, bevor sie von dem konturklassifizierenden Kalman-Filter verarbeitet werden.
- Zustandsgrößen, die nicht durch die Sensoren bestimmt werden oder als nicht zuverlässig genug bewertet wurden, sollen geschätzt werden.
- Objekthypothesen unterschiedlicher Sensoren sollen zu einer Objekthypothese zusammengefasst werden, so dass diese das gleiche Objekt beschreiben (ähnliche Position und Bewegungsinformation).

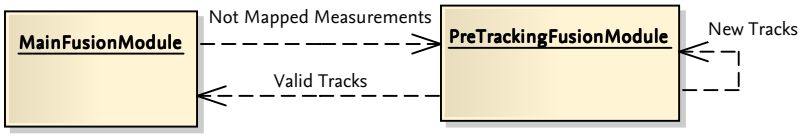


Abbildung 14.14: Komponentendiagramm bestehend aus *PreTrackingFusionModule* und *MainFusionModule*

Das *PreTrackingFusionModule* behandelt Messwerte, die vom *MainFusionModule* nicht zu bekannten Tracks zugeordnet werden konnten (siehe Abbildung 14.14). Auf diese Weise erzeugen die Messwerte nicht sofort neue Tracks innerhalb des *MainFusionModules*, sondern werden zunächst auf ihre Stabilität hin untersucht. Hierbei wird eine verzögerte Detektion der Objekthypothese zugunsten einer höheren Datenqualität in Kauf genommen. Der Fusion liegt der Gedanke zugrunde, dass, wenn sich eine Objekthypothese im *PreTrackingFusionModule* erfolgreich tracken lässt, die Güte der Daten ausreichend ist für das *MainFusionModule*. Hierzu wird eine adaptive *Mapping*-Komponente mit einer *Filter*-Komponente bei variablem Geometriemodell eingesetzt. Wird eine Objekthypothese über einen gewissen Zeitraum erfolgreich verfolgt, so wird ein Track im *MainFusionModule* erzeugt. Da die weiteren Messwerte des Objekts nun im *MainFusionModule* zugeordnet werden können, wird der Track des *PreTrackingFusionModules* nach einiger Zeit entfernt.

14.2.1 Eingesetzte Objekthypothesenmodelle

Das *PreTrackingFusionModule* hat u. a. die Aufgabe, fehlende und unzuverlässige Zustandsgrößen der Messdaten zu schätzen, damit ein verlässliches Tracking im *MainFusionModule* von Beginn an möglich ist. Den größten Einfluss auf die Qualität der Schätzung bei nicht stationären Objekten hat die Geschwindigkeitsinformation. Würde eine Objekthypothese mit einem falschen Geschwindigkeitsvektor initialisiert werden, so würde dies mit großer Wahrscheinlichkeit zum Verlust des Tracks führen. Mithilfe des CV-Modells kann diese Information für alle Tracks mithilfe der Messdaten der Sensoren bestimmt werden. Aus diesem Grund wird es im *PreTrackingFusionModule* eingesetzt.

Ein spezielles geometrisches Modell wird nicht gewählt, da das *PreTrackingFusionModule* keine Konturschätzung vornimmt. Aus diesem Grund richtet sich das Geometriemodell nach dem Modell des letzten Messdatums und wird bei jeder Aktualisierung ersetzt.

14.2.2 Adaptiver Zuordnungsalgorithmus

Die adaptive *Mapping*-Komponente bildet das Herzstück des Trackings zur Stabilitätsanalyse. Sie enthält zwei unabhängige Abschnitte. Der erste ordnet Objekthypothesen zu Tracks aufgrund der durch die Sensoren vergebenen Identifikationsnummer zu. Objekthypothesen, die auf diese Weise nicht verarbeitet werden konnten (z. B. bei der ersten Messung), werden durch einen lokalen Nearest-Neighbour-Algorithmus (Blackman u. Popoli, 1999, Seite 9f) zugeordnet. Dieser Algorithmus bestimmt für jede Objekthypothese einen Ähnlichkeitswert mit jedem Track (siehe Programmabschnitt 14.4). Der Track mit der höchsten Übereinstimmung gilt als zugeordnet. Wird ein gewisser Wert nicht überschritten, so findet keine Zuordnung statt. Diese Schwelle ε wird aufgrund der aktuellen Zuordnungskosten adaptiert.

Zur Anpassung des Schwellwerts wird ein Exponential Backoff Algorithmus genutzt (NORM IEEE Std 802.3, 2008, Seite 78). Hierbei wird die Differenz zwischen den für die Zuordnung

```
input: Measurements[], Tracks[], Gate
output: Gate, Mapping[]
foreach h ∈ Measurements do
  foreach t ∈ Tracks do
    if h is assigned to t
      Mapping.append(Track, Measurement)
    fi
  od
od
Cost = 0;
foreach h ∈ Measurements \ AlreadyMappedMeasurements do
  MaxValue = 0;
  SelectedTrack = undef
  foreach t ∈ Tracks do
    CurrentValue = Metric(t,h);
    Cost += CurrentValue;
    if (CurrentValue > MaxValue && CurrentValue > Gate)
      SelectedTrack = t;
      MaxValue = CurrentValue;
    fi
  od
  Mapping.append(Track, Measurement)
of
Gate = AdaptGating(Cost, Measurements.length, Gate);
```

Programmabschnitt 14.4: Pseudocode des adaptiven Zuordnungsalgorithmus

```
input: Cost, Measurements.length, ε
output: ε
if (Cost − ε ∗ Measurements.length ∗ 1.1) > 0
  ε ∗= 1.1;
else if (Cost − ε ∗ Measurements.length ∗ 0.95) < 0
  ε ∗= 0.95
fi
ε = max(ε, ε^Min);
ε = min(ε, ε^Max);
```

Programmabschnitt 14.5: Adaption der Schwellwerte des Zuordnungsalgorithmus (AdaptGating)

```
input: Polyline[]
output: TrackablePoint
if Polyline.length = 1
  TrackablePoint = Polyline[0];
else
  Rect = BoundingRect(Polyline)

  MinDistance = max
  for i=0;i<Polyline.length-1;i++ do
    P1 = Polyline[i];
    P2 = Polyline[i+1];
    ThisDistance = |P1P2, EgoPosition|_2;
    if ThisDistance < MinDistance
      TrackablePoint = FootOfPerpendicular(P1,P2,EgoPosition);
    fi
  od
  ClampToRect(Rect,TrackablePoint);
fi
```

Programmabschnitt 14.6: Berechnung des *TrackablePoints*

der aktuellen Messwerte benötigten Kosten und den maximal für diese Zuordnung erlaubten Kosten gebildet (siehe Programmabschnitt 14.5). Wird diese um mehr als 10 % überschritten, so wird der Schwellwert um 10 % erhöht. Werden die maximalen Kosten ε^{Max} nicht voll ausgeschöpft, so wird der Schwellwert verkleinert bzw. andersherum.

Als Metrik wird der euklidische Abstand genutzt. Da die Sensoren, wie in Abschnitt 14.1.4 beschrieben, sehr unterschiedliche geometrische Objekthypothesenmodelle nutzen, müssen diese Modelle vorverarbeitet werden. Hierzu werden sie auf einen einzelnen Punkt reduziert. Dieser *TrackablePoint* hat sich in der Praxis als verhältnismäßig stabil gegenüber Konturänderungen sowie Konturrauschen gezeigt.

Dabei wird zunächst unterschieden zwischen Objekthypothese mit einem Konturpunkt und mit zwei und mehr Stützpunkten (siehe Programmabschnitt 14.6). Für den Fall mehrerer Punkte wird zunächst das umschließende Rechteck gebildet. Anschließend wird für jeden Streckenabschnitt der kürzeste Abstand zur Position des Versuchsträgers bestimmt. Für den Abschnitt mit dem geringsten Abstand wird der Fußpunkt des Lots berechnet und dieser letztlich auf das einschließende Rechteck begrenzt (siehe Abbildung 14.15).

14.2.3 Kalman Filter

Als *Filter*-Komponente kommt ein Kalman-Filter zum Einsatz. Dieses schätzt die Geschwindigkeit, verändert das Geometriemodell jedoch nur für die Prädiktion von Objekthypothesen. Auf Basis des in Abschnitt 14.2.1 beschriebenen Objekthypothesenmodells ergibt sich der

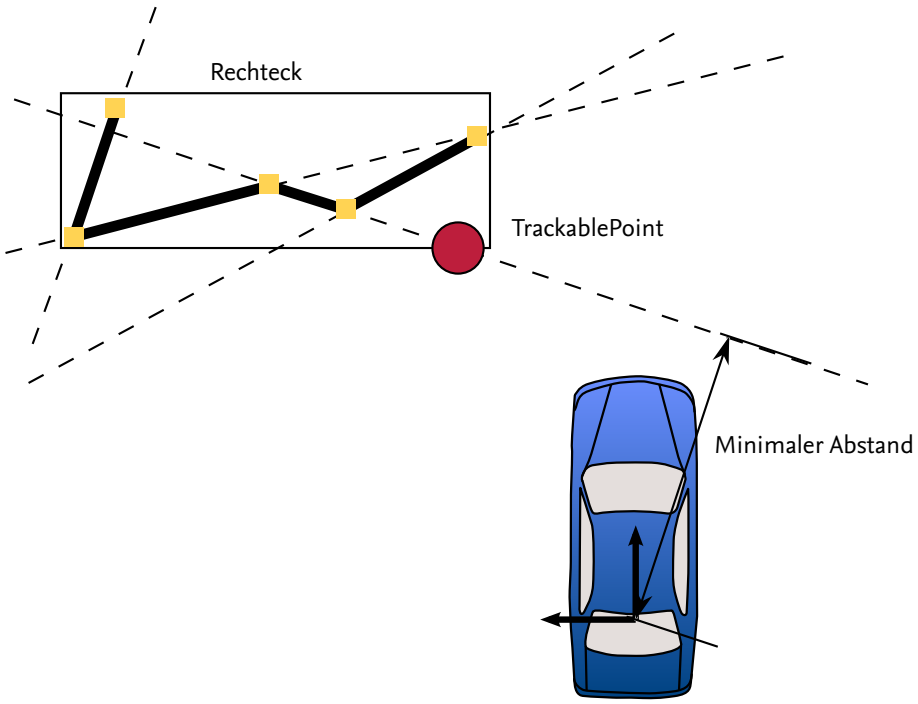


Abbildung 14.15: Berechnung des *TrackablePoint*

Zustandsvektor aus dem *TrackablePoint* und dem CV-Modell zu:

$$x_{\hat{PT}}[k] = \begin{bmatrix} TP_x[k] \\ TP_y[k] \\ v_x[k] \\ v_y[k] \end{bmatrix} \tag{14.48}$$

$$x[k+1] = TP_x[k] + \Delta t[k] \cdot v_x[k] \tag{14.49}$$

$$y[k+1] = TP_y[k] + \Delta t[k] \cdot v_y[k] \tag{14.50}$$

$$v_x[k+1] = v_x[k] \tag{14.51}$$

$$v_y[k+1] = v_y[k] \tag{14.52}$$

Die zugehörige Systemmatrix lautet:

$$F^{PT}[k] = \begin{bmatrix} 1 & 0 & \Delta t[k] & 0 \\ 0 & 1 & 0 & \Delta t[k] \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14.53}$$

Zur Aktualisierung des Zustandsvektors wird der *TrackablePoint* der Messkontur herangezogen:

$$\begin{aligned} H^{PT}[k] &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \text{, bei } y[k] = [TP_x[k], TP_y[k]]^T \\ H^{PT}[k] &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{, bei } y[k] = [TP_x[k], TP_y[k], v_x[k], v_y[k]]^T \end{aligned} \tag{14.54}$$

14.3 Reduktion von Sensorgeistern

Die von den genutzten Sensoren erzeugten Daten weisen i. d. R. eine hohe Datengüte auf. Jede der erzeugten Objekthypothesen entspricht normalerweise einem für die Applikation relevanten Objekt. Bedingt durch die Messprinzipien der Sensoren treten vereinzelt Ausnahmen dieser Annahme auf. So reagiert der Radarsensor SMS UMMR 2010 auf große metallische Gegenstände in der Fahrbahndecke oder die Lasersensoren nehmen eine Bodenwelle als Hindernis wahr. Durch die Diversität der Sensormessprinzipen treten diese Unzulänglichkeiten jedoch meist nicht gleichzeitig auf. Aus diesem Grund können Messdaten redundanter Messprinzipien miteinander abgeglichen werden und so die Erzeugung von Sensorgeistern reduziert werden.

Als Sensorgeist wird im Folgenden verstanden:

> Ein Sensorgeist ist ein Messfehler oder ein durch einen Sensor wahrgenommenes Objekt, deren Objekthypothese die Fahraufgabe der Applikation beeinträchtigt.

Zur Reduzierung von Sensorgeistern lassen sich, wie in Abschnitt 9.2.1 beschrieben, die Schnittstellen der Komponente *Basic* eines objekthypothesenbasierten Fusionsmoduls nutzen. Objekthypothesen, die über die entsprechende Schnittstelle empfangen wurden, werden durch das Fusionsmodul in einen neuen Track konvertiert. Dieser Datenstrom kann überwacht und so die Erzeugung von neuen Tracks verhindert oder verzögert werden. Damit lassen sich vor allem zwei Typen von Sensorgeistern reduzieren: kurzzeitige Messfehler und Messprinzip bedingte Sensorgeister. Den kurzzeitigen Messfehlern kann durch eine Verzögerung bei der Erzeugung neuer Tracks begegnet werden. Geister aufgrund des Messprinzips werden vor allem durch den Abgleich eines Messwerts mit den Erfassungsbereichen anderer Sensoren reduziert.

Zur Überwachung des Datenstroms wird die Komponente *NewTrackSelektion* eingesetzt. Sie wurde im Rahmen der Studienarbeit von Kählke implementiert (Kählke, 2010). Die Komponente empfängt alle durch das *PreTrackingFusionModule* aktualisierten Objekthypothesen und entscheidet anhand von logischen Ausdrücken über die Erzeugung von neuen Tracks im *MainFusionModule*.

Ob eine Objekthypothese zu einem Track wird, bestimmt sich durch die Zahl der Bestätigungen. Je nach Konfiguration sind mehr oder weniger Bestätigungen nötig. Hierzu können Ausdrücke in boolescher Logik (Bronstein u. a., 2001, Seite 353ff) definiert werden, welche die Existenz einer Objekthypothese in einem geschlossenen Polygonzug und die Messung durch einen Sensor auswerten.

```
input: LogicExpressions, Objecthypothesis
NeededUpdates = min;
for e ∈ LogicExpressions do
  if e.valid(Objecthypothesis) && NeededUpdates < e.Updates
    NeededUpdates = e.Updates;
  fi
od
if Objecthypothesis.Updates ≥ NeededUpdates
  Mark Objecthypothesis for Deletion;
  Create New Track;
else
  Objecthypothesis.Updates++;
fi
```

Programmabschnitt 14.7: Pseudocode des Algorithmus der Komponente *NewTrackSelektion*

Die Elemente der booleschen Logik umfassen, neben den Operationen $\vee, \wedge, \neg$ und den Elementen 1 und 0, die Zugehörigkeit einer Objekthypothese zu dem Erfassungsbereich eines Sensors (geschlossener Polygonzug) sowie die Existenz der mindestens einmaligen Messung durch einen Sensor. Auf diese Weise lassen sich Ausdrücke bilden wie:

$$(Erfassungsbereich1 \wedge Sensor1) \wedge (Erfassungsbereich2 \wedge Sensor2)$$

Wird der Ausdruck auf die Konfiguration aus Abbildung 14.16 angewendet, so trifft er für die rote Objekthypothese zu, auf die gelbe jedoch nicht.

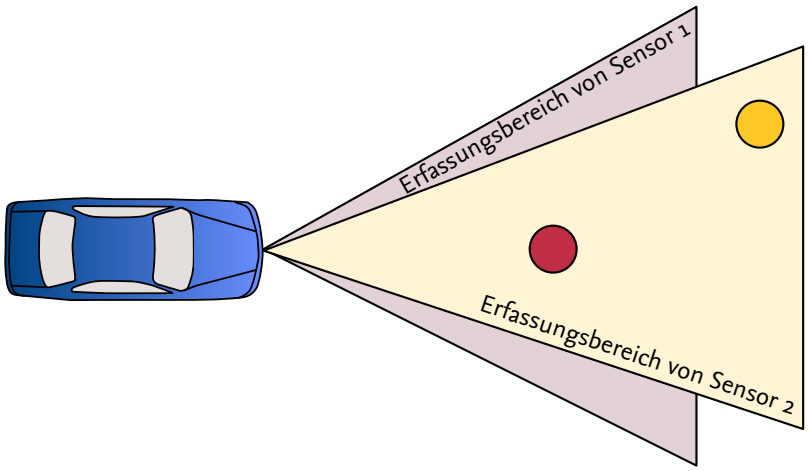


Abbildung 14.16: Beispielkonfiguration eines Versuchsträgers mit zwei Sensoren und zwei Objekthypothesen (rot und gelb)

Der Algorithmus evaluiert alle booleschen Ausdrücke und wählt den Ausdruck mit der maximalen Anzahl an notwendigen Aktualisierungen aus (siehe Programmabschnitt 14.7). Ist dieses Maximum erreicht, so wird die Objekthypothese im *PreTrackingFusionModule* als gelöscht markiert und ein neuer Track im *MainFusionModule* erzeugt. Zur Auswertung

```
<!DOCTYPE Configuration [
<!ELEMENT Condition(count, condition)>
<!ELEMENT count(#PCDATA)>
<!ELEMENT condition(Type, Element1, Element2, condition1, condition2,
    Modificator)>
<!ELEMENT Type('AND','OR','NOT')>
<!ELEMENT Element1(IDREF)>
<!ELEMENT Element2(IDREF)>
<!ELEMENT condition1(Type, Element1, Element2)>
<!ELEMENT condition2(Type, Element1, Element2)>
<!ELEMENT Modificator(#PCDATA)>
<!ELEMENT SensorBoundaryPolygons(Name, Polygon)>
<!ELEMENT Name(ID)>
<!ELEMENT Polygon(#PCDATA)>
<!ELEMENT SensorIds(ID)>
]>
```

Programmabschnitt 14.8: Dokumenttypedefinition der Datenstruktur zur Beschreibung von redundanten Erfassungsbereichen der genutzten Sensoren, Erweiterung nach Kählke (2010, Seite 21)

der Ausdrücke muss zunächst die Zugehörigkeit der Objekthypothese zu den einzelnen Polygonzügen bestimmt werden. Darüber hinaus ist die Information über die Existenz einer Messung eines Sensors für eine Objekthypothese nötig. Diese ist in der Datenstruktur *UnifiedSensorObject* jedoch nicht vorhanden. Hierzu stellt die Instanz der *Filter*-Komponente des *PreTrackingFusionModules* eine eigene Schnittstelle speziell für dieses Modul bereit.

Der Algorithmus benötigt das Wissen über die Sensorkonfiguration sowie die Regeln zur Entscheidung über die Erstellung von neuen Tracks. Hierzu wird eine XML-Datenstruktur genutzt (siehe Programmabschnitt 14.8). Sie enthält die Menge der vorkommenden Sensoridentifikationsnummern, die Polygonzüge der Erfassungsbereiche der Sensoren sowie die booleschen Ausdrücke. Die enthaltenen Regeln der Konfiguration für das Projekt Stadtpilot wurden anhand der Dokumentation der Sensoren sowie durch Messungen der realen Erfassungsbereiche der Sensoren durchgeführt. Die Anzahl der notwendigen Aktualisierungen sowie die Nutzung der Redundanzen ist empirisch ermittelt worden.

15 Gitterbasierte Fusion mit Bayes-Filter

Innerhalb des Projekts Stadtpilot wurde das gitterbasierte Fusionsmodul im Rahmen der Studienarbeit von Müller umgesetzt (Müller, 2012). Dieses verarbeitet die Messdaten von unterschiedlichen Sensoren und wird über die *View*-Komponenten *DrivingTubeView*, *OcclusionView* und *ObjectReconstructionView* ausgewertet.

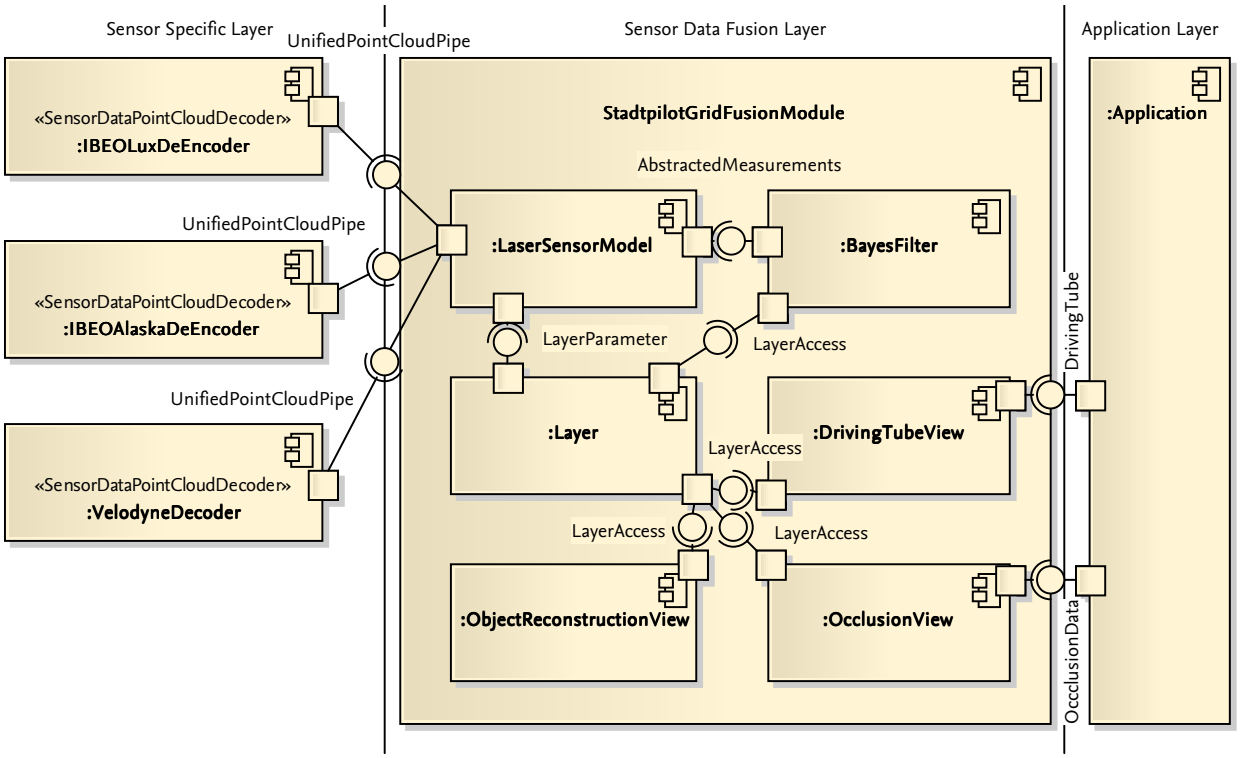


Abbildung 15.1: Datenfluss und Aufteilung der gitterbasierten Fusionsmodule im Projekt Stadtpilot. Die Typen der Schnittstellen zwischen den Komponenten finden sich in Abschnitt 9.2.2. Sie sind aus Gründen der Übersichtlichkeit nicht mit abgebildet.

In Abbildung 15.1 wird der Datenfluss der am gitterbasierten Fusionsmodul beteiligten Komponenten dargestellt. Innerhalb des *Sensor Specific Layers* werden die Daten der Sensoren IBEO Alaska XT, IBEO Lux und Velodyne HDL64ES2 dekodiert. Die durch die Dekodierung entstehenden Objekte der Klasse *UnifiedPointCloud* stellen die Eingangsgröße für das *StadtpilotGridFusionModule* dar. Da es sich bei allen Sensoren um Lasersensoren mit ähnlichen Messprinzipien handelt, werden sie durch dasselbe *LaserSensorModel* verarbeitet und anschließend durch ein binäres Bayes-Filter fusioniert.

Als Ergebnis der Datenfusion können mit den *View*-Komponenten drei der projektspezifischen Anforderungen erfüllt werden. Der *OcclusionView* bestimmt den aktuellen durch Verdeckungen eingeschränkten Erfassungsbereich der Sensoren (Anforderung SPFA29).

Durch den *ObjectReconstructionView* werden vor allem statische Objekte der Fahrzeugumgebung wahrgenommen und in *UnifiedSensorObjects* konvertiert, welche wiederum durch das *MainFusionModule* verarbeitet werden können (Anforderung SPFA31). Der letzte *View* bestimmt den befahrbaren Bereich in der geplanten Route des Versuchsträgers (Anforderung SPFA26).

15.1 Verarbeitung von Lasersensordaten

Das in diesem gitterbasierten Fusionsmodul genutzte *SensorModel* verarbeitet Daten von Lasersensoren. Diese werden durch die Datenstruktur *UnifiedPointCloud* abgebildet, welche eine Liste von 3D-Messpunkten in Weltkoordinaten enthält. Die Grundannahme zur Verarbeitung dieser Messpunkte ist, dass vom Ursprung des Sensors kein anderes Objekt den Pfad des Laserstrahls blockiert und damit für den Versuchsträger befahrbar ist.

Das implementierte Sensormodell bildet eine sehr einfache Modellvorstellung ab. Hierbei wird für Zellen vom Sensorursprung zum Reflektionspunkt die Wahrscheinlichkeit p_{free} und für die Zelle des Reflektionspunkts der Wahrscheinlichkeit $p_{\overline{free}}$ angenommen.

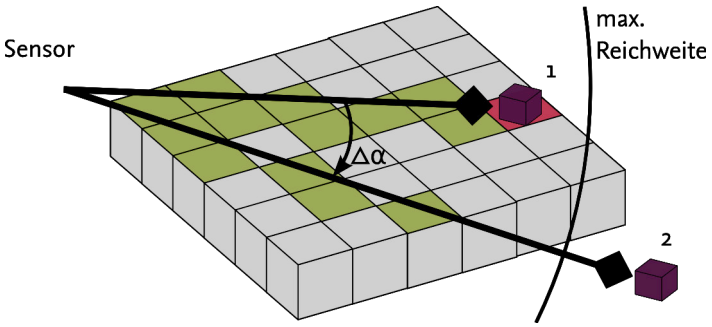


Abbildung 15.2: Abbildung von Messpunkten außerhalb der maximalen Reichweite eines Sensors. Wahrscheinlichkeit p_{free} (grün), Wahrscheinlichkeit $p_{\overline{free}}$ (rot), Objekt 1 innerhalb der Gitterdatenstruktur, Objekt 2 außerhalb der Gitterdatenstruktur

Die eingesetzten Sensoren messen Reflektionen ihres Lasers nur bis zu einer gewissen Reichweite. Ist diese Reichweite überschritten, so kann kein Entfernungswert bestimmt werden. Aus diesem Grund sind diese „Nicht-Messpunkte“ auch nicht Bestandteil der Punktwolke der *UnifiedPointCloud*. Sie können jedoch mithilfe des Richtungsvektors, der Winkelauflösung $\Delta\alpha$ sowie der maximalen Reichweite r des Sensors rekonstruiert werden (siehe Abbildung 15.2). Dabei werden die Richtungsvektoren aller möglichen Messpunkte bestimmt, diskretisiert und anschließend sortiert. Sind die Elemente der sortierten Liste nicht äquidistant verteilt, so stellen die fehlenden Elemente Messungen ohne Reflexion dar.

Der IBEO Alaska XT löst jedoch Messungen je nach Messrichtung mit unterschiedlichen Differenzwinkeln auf, welche berücksichtigt werden müssen (siehe Abbildung 15.3). Direkt in Richtung des Sensorkoordinatensystems werden die Messdaten mit $0,125°$, zu den Seiten jedoch nur noch mit $0,5°$ aufgelöst. In Programmabschnitt 15.1 wird der Algorithmus zusammengefasst. Zunächst werden für eine *UnifiedPointCloud* die Richtungsvektoren berechnet und sortiert. Darauf folgend wird über die Liste der Messpunkte iteriert. Tritt eine Lücke zwischen zwei Messwerten auf, so werden die Zellen in dieser Richtung mit dem Wert p_{free} abstrahiert. Für die Zelle des Messwerts wird der Wert $p_{\overline{free}}$ angenommen.

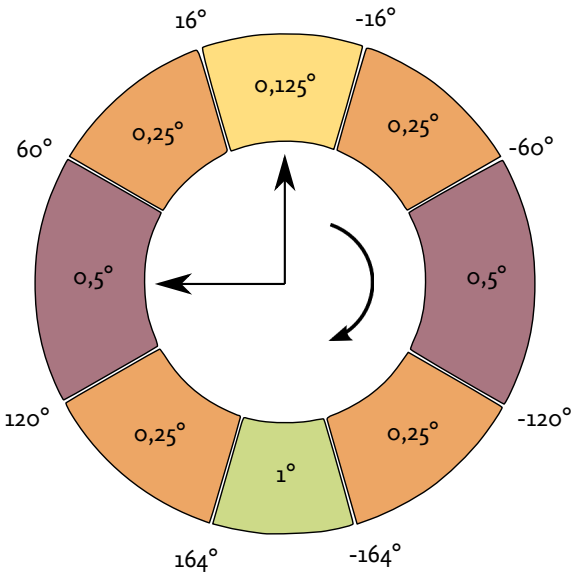


Abbildung 15.3: Winkelauflösung des IBEO Alaska XT bei 12,5 Hz Rotationsfrequenz (Ibeo Automobile Sensor GmbH, 2006, Seite 12)

```
input: UnifiedPointCloud, Resolutiontable
output: AbstractedCells[]
foreach p ∈ UnifiedPointCloud do
  CalculateDiscreteDirection(p)
od
SortByDirection(UnifiedPointCloud);
LastDirection = 0;
foreach p ∈ UnifiedPointCloud do
  while p.Direction−LastDirection > GetResolution(Resolutiontable,
     LastDirection) do
    LastDirection += GetResolution(Resolutiontable, LastDirection);
    EmptyRay = [r∗cos(LastDirection) r∗sin(LastDirection)]
    while (c = Bresenham(EmptyRay)) do
      AbstractedCells.add(c, p_free);
    od
  od
  while (c = Bresenham(p)) do
    AbstractedCells.add(c, p_free);
  od
  AbstractedCells.add(p, p_¬free);
od
```

Programmabschnitt 15.1: Pseudocode des SensorModells zur Verarbeitung von Laserdaten

15.2 Fusionsalgorithmus

Durch das *SensorModel* wurden eine Reihe von Gitterzellen entlang des Strahlengangs eines Laserimpulses zur Aktualisierung ausgewählt. Für jede dieser Zellen existiert eine Belegtheitswahrscheinlichkeit $p[k]^{Cell}$, die es mit einer aus dem aktuellen Messwert resultierenden Wahrscheinlichkeit $p[k]^{Model}$ zu kombinieren gilt. Hierzu wird im *StadtpilotGridFusionModule* als Fusionsalgorithmus ein binäres Bayes-Filter genutzt:

$$p^{Cell}[k+1] = \frac{p[k]^{Model} \cdot p[k]^{Cell}}{p[k]^{Model} \cdot p[k]^{Cell} + (1.0 - p[k]^{Model}) \cdot (1.0 - p[k]^{Cell})} \quad (15.1)$$

$$p^{Cell}[k] \in (0,1) \quad (15.2)$$

$$p^{Model}[k] \in (0,1) \quad (15.3)$$

Damit Formel 15.1 Gültigkeit erlangt, wird ein Markov Prozess 1. Ordnung angenommen, sowie die einzelnen Zellen als statistisch unabhängig betrachtet. Die Herleitung kann in Elfes (1990) nachvollzogen werden.

15.3 Nachführung der Gitterstruktur

Eine Realisierung der *Modifier*-Komponente ist die Nachführung der Gitterstruktur mit dem Versuchsträger. Die eingesetzte Gitterstruktur im Projekt Stadtpilot deckt eine ortsfeste Fläche von $102.4m \times 102.4m$ bei einer Auflösung von $20cm \times 20cm$ um das Fahrzeug ab. Aufgeteilt ist der *Layer* in insgesamt 8 x 8 *GridBlocks à* 64 x 64 Zellen. Der Versuchsträger soll sich immer innerhalb der inneren 4 x 4 *GridBlock*-Gruppe befinden. Dies wird zyklisch überprüft und ggf. wird eine Verschiebung vorgenommen.

```
input: CenterOfLayer, EgoPosition
DistanceToCenter = CenterOfLayer − EgoPosition;
DeltaX = 0;
if |DistanceToCenter.X| > Layer.BlockSizeX
  DeltaX = ⌊DistanceToCenter.X / Layer.BlockSizeX + 0.5⌋;
fi
DeltaY = 0;
if |DistanceToCenter.Y| > Layer.BlockSizeY
  DeltaY = ⌊DistanceToCenter.Y / Layer.BlockSizeY + 0.5⌋;
fi
if DeltaX ≠ 0 || DeltaY ≠ 0
  Layer.Shift([DeltaX, DeltaY]);
fi
```

Programmabschnitt 15.2: Pseudocode der Nachführung eines Layers

In Programmabschnitt 15.2 ist der Algorithmus zusammengefasst. Dabei wird zunächst der Abstand des Versuchsträgers zum Zentrum der Gitterstruktur berechnet. Ist dieser Abstand in X- oder Y-Richtung größer als ein *GridBlock*, so wird eine Verschiebung um die entsprechende Anzahl an Blöcken vorgenommen.

15.4 Schätzung der Fahrbahnbreite

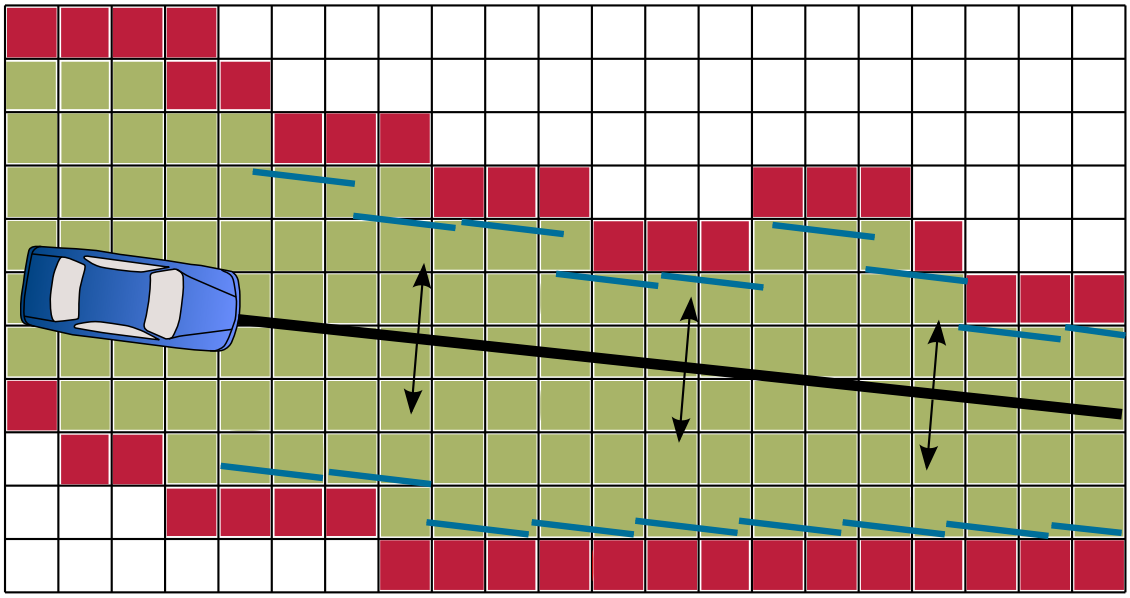


Abbildung 15.4: Algorithmus zur Schätzung des Fahrkorridors nach Weiss (2011). Belegte Zellen (rot), nicht belegte Zellen (grün), Fahrtrichtung (schwarze Gerade), geschätzte Fahrbahnbreite (blaue Geraden)

In Weiss (2011, Seite 128ff) wurde ein Algorithmus zur Schätzung des Fahrkorridors beschrieben. Dieser schätzt den aktuellen befahrbaren Bereich vor dem Fahrzeug auf Basis einer Gitterdatenstruktur. Hierzu wird eine Gerade in Fahrtrichtung des Versuchsträgers angenommen und diese abschnittsweise in beide Richtungen lotrecht dazu verschoben. Diese Verschiebung wird durch belegte Zellen gestoppt (siehe Abbildung 15.4). Auf diese Weise erhält man einen diskretisierten linken und rechten Fahrbahnrand.

Dieser Algorithmus wurde an die Möglichkeiten des Projekts Stadtpilot adaptiert und als Implementierung einer *View*-Komponente umgesetzt. In der Anforderung SPFA26 wurde festgestellt, dass die in Abschnitt 3.6 beschriebene digitale Karte nicht immer dem aktuellen Verkehrsgeschehen entspricht. So sind beispielsweise andere Fahrzeuge so abgestellt, dass sie leicht in den Fahrstreifen ragen oder die Güte der Karte entspricht nicht durchgehend den Anforderungen der Applikation. In diesen Fällen wäre eine kurzzeitige Anpassung der geplanten Trajektorie des Versuchsträgers wünschenswert.

Da im Versuchsträger, im Vergleich zur Arbeit von Weiss, eine statische digitale Karte mit Fahrbahnverlauf vorliegt, kann diese genutzt werden, um den Suchbereich des Algorithmus vorzugeben. Mit dieser Erweiterung ist das Verfahren nicht länger auf eine korrekte Ausrichtung des Versuchsträgers angewiesen und ist in der Lage, auch kurvige Fahrstreifenabschnitte korrekt abzubilden. Darüber hinaus wird neben belegten Gitterzellen die aus der Karte bekannte Fahrstreifenbreite als Abbruchkriterium bei der Verschiebung genutzt.

Abbildung 15.5 stellt das veränderte Verfahren dar. Abgebildet ist eine kurvige Strecke, in der ein abgestelltes Fahrzeug den Fahrstreifen einschränkt. Meist werden die verschobenen Trajektorienabschnitte durch die Fahrstreifenbegrenzungen aus der digitalen Karte

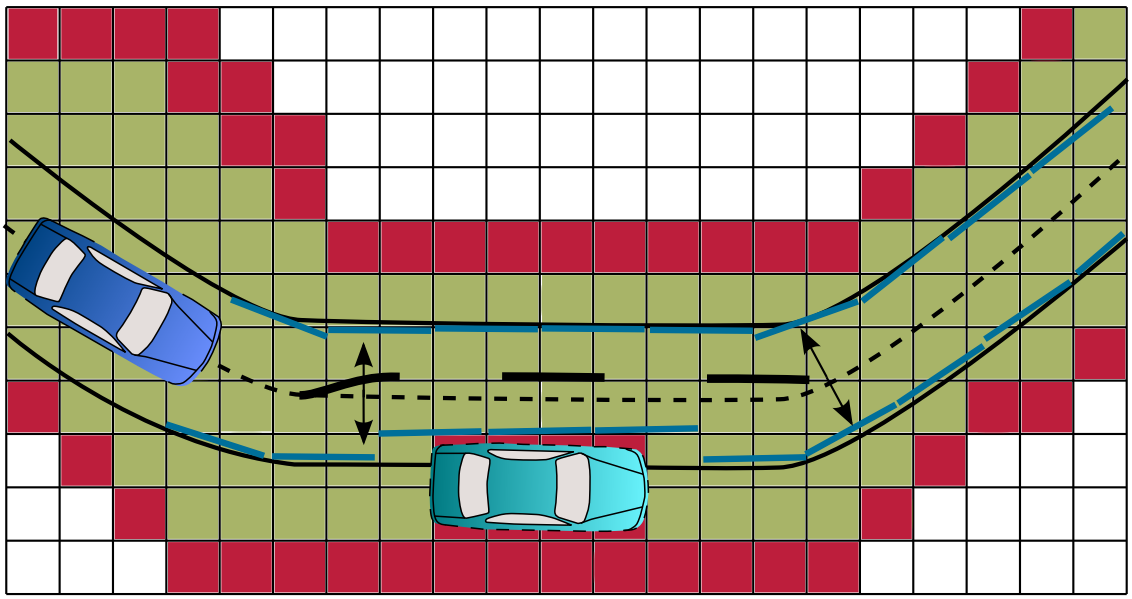


Abbildung 15.5: Algorithmus zur Anpassung der Fahrstreifenbreite. Belegte Zellen (rot), nicht belegte Zellen (grün), ursprüngliche Trajektorie (klein gestrichelte Linie), veränderte Trajektorie (grob gestrichelte Linie), geschätzte Fahrbahnbreite (blaue Geraden)

beschränkt. Das abgestellte Fahrzeug schränkt diesen Bereich jedoch weiter ein, sodass die durch den Fahrentscheider des Versuchsträgers bisher geplante Trajektorie geringfügig angepasst werden muss, um nicht mit dem anderen Fahrzeug zu kollidieren.

15.5 Bestimmung des aktuellen Erfassungsbereiches der Sensoren

Um eine Entscheidung über bestimmte Manöver (z. B. Fahrstreifenwechsel oder Abbiegen durch fließenden Verkehr) zu treffen, ist das Wissen über die Verfügbarkeit von aktuellen Messdaten eine wichtige Größe (Anforderung SPFA29). Zur Umsetzung der entsprechenden Anforderung wurde eine weitere Implementierung der *View*-Komponente realisiert (Ulbrich, 2011, Seite 37).

Diese Komponente bestimmt einen Wert für die Beobachtbarkeit Ξ aus dem Wert der Zelle p^{Cell}, in dem dieser zum Abstand zur geplanten Trajektorie des Versuchsträgers in Relation gesetzt wird. Zur Realisierung müssen zunächst die Fahrstreifen in der digitalen Karte bestimmt werden, welche durch das eigene Fahrzeug gekreuzt werden. Für den Bereich dieser Fahrstreifen wird nun dynamisch ein Layer erzeugt bzw. an diese Stelle verschoben und in der Größe angepasst. Die Zellen innerhalb der Fahrstreifen werden nun durch die Daten der Sensoren aktualisiert und zyklisch ausgewertet.

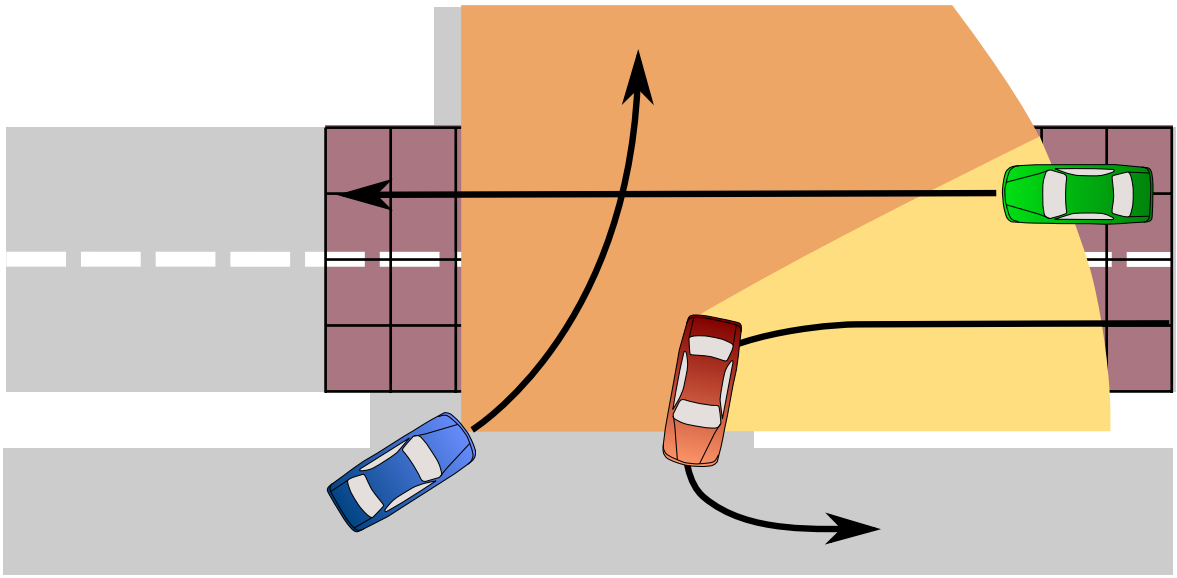


Abbildung 15.6: Abbiegesituation zweier Fahrzeuge. Gitterdatenstruktur (rot), durch Sensor erfasster Bereich (orange), nicht durch Sensor erfasster Bereich (gelb), eigenes Fahrzeug (blau), abbiegendes Fahrzeug (rot), verdecktes Fahrzeug (grün)

Hierzu werden die Zellen unter Berücksichtigung des Abstands zum Kreuzungsbereich zwischen Trajektorie und Fahrstreifen aufaddiert (siehe Abbildung 15.6). Relevant ist dabei die Konfidenz in den Wert einer Zelle. Letztlich wird die Summe über alle Zellen normiert:

$$d = |[x_i^{Cell} y_i^{Cell}] - [x^{ConflictPoint} y^{ConflictPoint}]|_2 \tag{15.4}$$

$$\Upsilon = \sum_{i \in CellsOfConflictingLanes} \frac{2 \cdot \lfloor p_i^{Cell} - \frac{1}{2} \rfloor}{d} \tag{15.5}$$

$$\Lambda = \sum_{i \in CellsOfConflictingLanes} \frac{1}{d} \tag{15.6}$$

$$\Xi = \frac{\Upsilon}{\Lambda} \tag{15.7}$$

15.6 Konstruktion von Objekthypothesen aus Gitterdaten

In einer ortsfesten Gitterdatenstruktur bilden sich ortsfeste Objekte gut heraus. Diese Stärke wird durch eine weitere Instanz der *View*-Komponente genutzt, welche Anforderung SPFA31 umsetzt. Hierbei werden aus den Gitterdaten Objekthypothesen mit einem Kontur-Geometriemodell generiert, die anschließend dem *MainFusionModule* zugeführt werden könnten. Auf diese Weise können die beiden Fusionsformen kombiniert und die Daten der objekthypothesenbasierten Fusion gestützt werden.

Zur Realisierung wurde ein 2D-Split&Merge-Algorithmus entwickelt, mit dem sich Gitterdatenstrukturen effizient auswerten lassen. Der Algorithmus wurde in der Studienarbeit von Liu in Matlab umgesetzt (Liu, 2012).

```
input: S, E_max
output: S
do
  while E > E_max do
    Find Segment S_i with E_i > E_j ∀j ∈ 1...n, j ≠ j;
    Split S_i;
  fi
  while E < E_max do
    Find Segments S_i, S_j, S_i ∪ S_j results in minimum increase of E;
  od
  Adjust Line Endpoints;
while(Something changed)
```

Programmabschnitt 15.3: Pseudocode eines Split&Merge-Algorithmus nach Pavlidis u. Horowitz (1974)

Traditioneller Split&Merge-Algorithmus

Split&Merge-Algorithmen finden in der Robotik eine weite Verbreitung (z. B. Borges u. Aldon (2004)) oder Nguyen u. a. (2007)). In ihrer traditionellen Form (Pavlidis u. Horowitz, 1974) wird diese Algorithmen-Gruppe in der Robotik vor allem zur Auswertung von scannenden 1D-Lasersensoren eingesetzt (Borges u. Aldon, 2004). Diese Sensoren messen den Abstand zu einem den Laserstrahl reflektierenden Objekt in Polarkoordinaten (r, α) und geben diesen Wert zur Verarbeitung an den Algorithmus weiter.

Dabei lösen die Algorithmen folgendes Optimierungsproblem:

> „Für eine Menge an Punkten $S = \{x_i, y_i | i = 1, 2 \dots N\}$ bestimme die minimale Zahl n, sodass S in n Untergruppen $S_1, S_2 \dots S_n$ aufgeteilt werden kann, wobei die Punkte jeder Untergruppe durch ein Polynom des Grades $m - 1$ mit einem kleineren Fehler als e bezüglich einer Fehlernorm abgebildet werden kann.“ (Übersetzung nach Pavlidis u. Horowitz (1974, Seite 860))

Zur Realisierung der Rekonstruktion von Objekthypothesen mit dem Polyline-Geometriemodell werden Polynome zweiten Grades angenommen.

Der in Programmabschnitt 15.3 beschriebene abstrakte Algorithmus lässt sich zur Verarbeitung von Laserdaten anpassen. Dazu wird die Ordnung der Messpunkte genutzt, sodass diese jeweils nur einmal untersucht werden. Die Anzahl der zu untersuchenden Segmente reduziert sich dabei auf eins und es muss jeweils nur entschieden werden, ob der aktuelle Messpunkt dem aktuellen Segment hinzugefügt oder ein neues erstellt wird. Im zweiten Schritt werden die Segmente zu Konturen zusammengefasst.

Abbildung 15.7 zeigt das Ergebnis der Verarbeitung von Messdaten eines Laserscanners. Zu sehen sind die Messpunkte (rot), die von dem Sensor aufgenommen wurden. Diese werden sequenziell verarbeitet. Beginnend im Uhrzeigersinn trifft der Laser auf ein Objekt und beginnt das erste Segment S_1. Daran anschließend werden Messpunkt zwei und drei ebenfalls diesem Segment zugeordnet, da sie sich durch eine Gerade gut approximieren lassen. Messstrahl vier trifft nicht auf ein Objekt und beendet damit das Segment S_1. Die Messpunkte fünf bis sieben werden entsprechend dem Segment S_2 zugeordnet.

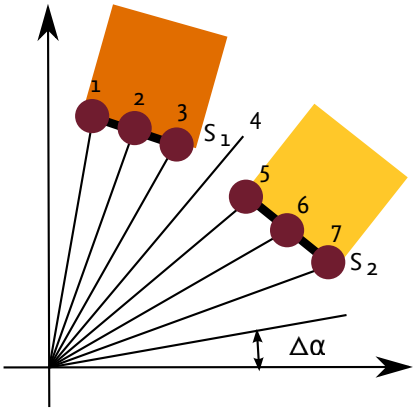


Abbildung 15.7: Segmentierte Messdaten eines rotierenden Lasersensors. Messpunkte (rot)

Zur Wahrnehmung der Unterbrechungspunkte zwischen zwei Segmenten beschreiben Borges u. Aldon beispielsweise die folgenden zwei Varianten (Borges u. Aldon, 2004):

Adaptive Breakpoint Detector: Dieser sehr einfache Algorithmus basiert auf dem Abstand zwischen zwei Messpunkten. Ist dieser größer als der Wert ε, so wird ein neues Segment begonnen. Da, bedingt durch das Messverfahren eines rotierenden Lasersensors, die Messwerte in Polarkoordinaten vorliegen, muss berücksichtigt werden, dass Messpunkte die weiter vom Sensorursprung entfernt liegen, in einem größeren Abstand zum nächsten Messpunkt resultieren.

$$p_n = \begin{bmatrix} r_n \cdot \cos\alpha_n & r_n \cdot \sin\alpha_n \end{bmatrix} \tag{15.8}$$

$$\varepsilon = r_{n-1} \cdot \frac{\sin\Delta\alpha}{\sin\alpha - \Delta\alpha} + 3\sigma \tag{15.9}$$

$$|p_n - p_{n-1}|_2 > \varepsilon \Rightarrow \text{ neues Segment} \tag{15.10}$$

Kalman-Filter Breakpoint Detector: Mithilfe eines Kalman-Filters lassen sich Unterbrechungspunkte auf der Basis von stochastischen Überprüfungen beschreiben. Die Suche nach Unterbrechungspunkten wird dabei durch das Modell einer Geraden beschrieben. Verletzt ein Messpunkt dieses Modell, so ist ein Unterbrechungspunkt gefunden. Der Zustandsvektor entspricht dabei $\hat{x}[k] = \begin{bmatrix} r[k] & \frac{dr[k]}{d\alpha[k]} \end{bmatrix}^T$. Die Zustandsübergangsmatrix $F[k]$ sowie die Abbildung der Messwerte $H[k]$ lauten:

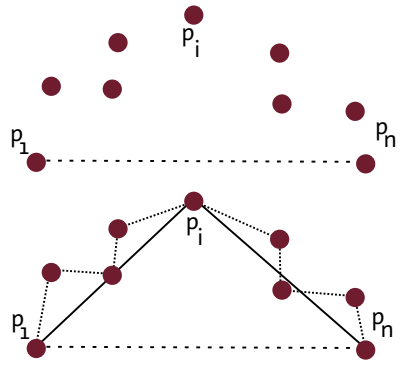
$$F[k] = \begin{bmatrix} 1 & \Delta\alpha[k] \\ 0 & 1 \end{bmatrix} \tag{15.11}$$

$$H[k] = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{15.12}$$

Durch einen χ^2-Test (Brumback u. Srinath, 1987) kann anschließend eine Modellverletzung nachgewiesen werden.

$$\frac{\tilde{y}[k]^2}{S[k]} \geq 3.84\ (5\%) \Rightarrow \text{ neues Segment} \tag{15.13}$$

Die so gewonnenen Segmente werden im nächsten Schritt durch Geradenfunktionen der Form $f(x) = m \cdot x^1 + c \cdot x^0$ dargestellt. Hierzu werden in der Literatur unterschiedliche Methoden vorgeschlagen (Borges u. Aldon, 2004, Seite 280ff). Im Rahmen dieser Arbeit wird der Algorithmus Iterative End-Point-Fit (IEPF) genutzt.




```
input: ε, L[] = {p_1, p_n}
output: L[]
if |p_maxDist − p_1p_n|_2 > ε
  L = IEPF({p_i ∈ S|1 ≤ i < maxDist}, ε)
        ∪
      IEPF({p_i ∈ S|maxDist ≤ i ≤ n}, ε);
fi
```

Abbildung 15.8: Iterative End-Point-Fit. Algorithmus (rechts), Anwendung (links)

IEPF wird einerseits zur Segmentierung und andererseits zur Vereinfachung von Polygonzügen eingesetzt. Der Algorithmus betrachtet einen Polygonzug als Menge von sortierten Punkten $S_i = \{p_1 \dots p_n\}$. Zunächst werden die Punkte p_1 und p_n durch eine Gerade verbunden und anschließend der Punkt mit dem größten Abstand zu dieser Gerade bestimmt. Ist dieser Abstand größer als ein Wert ε, so wird die Punktmenge an diesem Punkt aufgespalten und für jede der beiden neuen Punktmengen die Prozedur wiederholt. In Abbildung 15.8 ist der Initialschritt sowie das Ergebnis des Algorithmus beispielhaft dargestellt. Die approximierte Kontur wird als durchgezogene Linie dargestellt und ist gegenüber der gepunkteten Linie deutlich vereinfacht worden.

2D-Split&Merge-Algorithmus

Die im Bereich der Robotik erfolgreich eingesetzten Split&Merge-Algorithmen eignen sich vor allem zur Auswertung von sortierten 1D-Messpunktlisten. Ist diese Voraussetzung nicht mehr gegeben, z. B. durch Messpunkte, die sich in einem 2D-Raum befinden, so ist die Annahme, dass zu jedem Zeitpunkt nur ein Segment betrachtet werden muss, nicht mehr erfüllt.

Darüber hinaus ermöglichen die am Stadtpilot-Versuchsträger befestigten Sensoren die Wahrnehmung von Objekten, die, aus Sicht des Versuchsträgers, durch andere Objekte verdeckt sind (siehe Abbildung 15.9). Durch eine 1D-Darstellung ließen sich diese Konturen ebenfalls nicht rekonstruieren.

Diese Einschränkung macht den Algorithmus zur Auswertung von Gitterdatenstrukturen nicht einsetzbar. Aus diesem Grund wurde hierzu eine Variante entwickelt, die 2D-Datenstrukturen verarbeiten und so die Objektkonturen rekonstruieren kann. Der Grundgedanke besteht darin, innerhalb des Split-Schrittes mehr als ein Segment gleichzeitig zu verarbeiten (siehe Programmabschnitt 15.4). Dabei wird zunächst mithilfe des Bresenham-Algorithmus (Bresenham, 1965) eine Menge an Zellen in Abhängigkeit eines Blickwinkels der Gitterstruktur bestimmt. Diese Zellmenge wird nun auf die Übereinstimmung einer belegten Zelle mit der aktuellen Geraden-Schätzung eines Kalman-Filters hin untersucht.

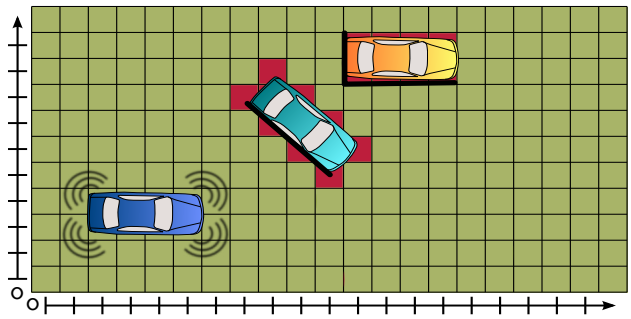


Abbildung 15.9: Nicht durch 1D-Split&Merge-Algorithmen abbildbare Situation. Belegte Zellen (rot), nicht belegte Zellen (grün)

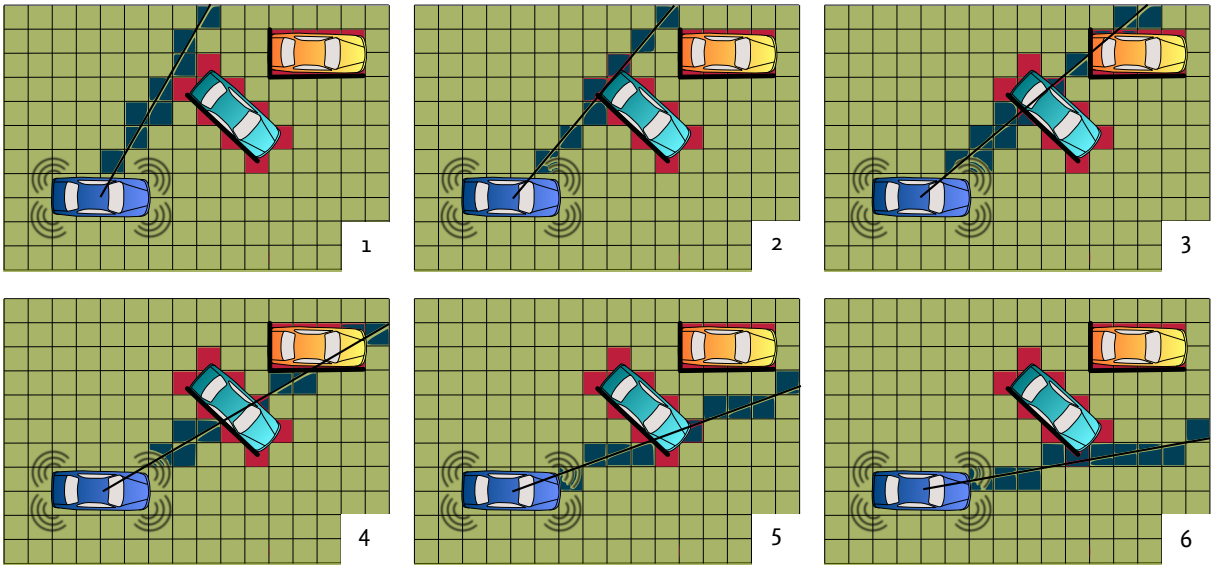


Abbildung 15.10: Schrittweise Darstellung des 2D-Split&Merge-Algorithmus. Belegte Zellen (rot), nicht belegte Zellen (grün), aktuell ausgewählte Zellen (blau)

Dabei wird das Kalman-Filter genutzt, dessen χ^2-Wert aktuell am niedrigsten ist und mit der Position der Zelle aktualisiert („Live“-Filter). Ist der χ^2-Wert dennoch größer als z. B. 5 %, so wird eine neue Gerade und damit auch ein neuer Kalman-Filter initialisiert. Nachdem alle Zellen abgearbeitet wurden, werden die Zustandsvektoren aller Kalman-Filter, die in diesem Durchlauf nicht genutzt wurden, als abgeschlossene Geraden angesehen. Diese „Dead“-Filter werden in den nächsten Durchläufen nicht mehr betrachtet, da sie nicht durch Messwerte gestützt werden konnten.

Darauf folgend werden die Segmente zu Polygonzügen zusammengefasst. Hierzu wird jedes bisher nicht behandelte Segment auf einen Schnitt mit dem aktuellen Segment untersucht. Ist dies der Fall, so werden beide Segmente kombiniert. Abschließend wird bei den erzeugten Konturen der IEPF-Algorithmus zur Vereinfachung angewandt.

In Abbildung 15.10 ist der Ablauf des ersten Teils des Algorithmus in sechs Schritten dargestellt. Beginnend oben links, wird eine Menge von Zellen durch den Bresenham

```
input: Cells
output Polylines[]
LastLiveKF = Segments  = [];
for α = 0;α < 2π;α+ = Δα do
  LiveKF = [];
  foreach c ∈ Bresenham(Cells,α)|c = occupied do
    [MinValue,CurrentKF] = min(ỹ[k]²/S[k], ∀LastLiveKF);
    if MinValue ≥ 3.84 // (5\%) χ² − Test
      CurrentKF = NewKF(c);
    else
      CurrentKF.Update(c);
    fi
    CurrentKF.Measurements.add(c)
    LiveKF.add(CurrentKF);
  od
  foreach k ∈ LastLiveKF \ LiveKF do
    Segments.add(k.Measurements);
  od
  LastLiveKF = LiveKF;
od
foreach s1 ∈ Segments| not s1.merged do
  p = [s1];
  foreach s2 ∈ Segments \ s1| not s2.merged do
    p.add(s2);
    s2.merged = true;
  od
  Polylines.add(p);
od
foreach p ∈ Polylines do
  IEPF(p);
od
```

Programmabschnitt 15.4: Pseudocode des 2D-Split&Merge-Algorithmus

Algorithmus selektiert. Auf dem Weg von Versuchsträger bis zum Rand der Gitterstruktur liegen keine belegten Zellen. Deshalb wird in diesem Schritt auch kein Kalman-Filter initialisiert. Im zweiten Schritt wird die Richtung um 10° gedreht und der Weg triff auf das erste Fahrzeug. Hierbei wird das erste Kalman-Filter erstellt. Im dritten Schritt trifft der Strahl zunächst auf das erste Fahrzeug. Es existiert zurzeit nur ein Kalman-Filter, dessen χ^2 Test positiv ausfällt. Das Filter wird also mit der Zelle aktualisiert und in die „Live"-Filterliste übernommen. Auf dem weiteren Weg werden die Zellen des zweiten Fahrzeugs berührt. Der χ^2 Test fällt hier negativ aus und ein zweites Kalman-Filter wird initialisiert.

Im vierten Schritt werden ebenfalls wieder beide Fahrzeuge durch den virtuellen Messstrahl berührt und die jeweiligen Filter aktualisiert. Der fünfte Schritt berührt nur noch das vordere Fahrzeug. Da hier nur ein Filter aktualisiert wurde, ist das Filter des hinteren Fahrzeugs im sechsten Schritt nicht mehr aktiv und zählt somit zu den „Dead"-Filtern. Der sechste

Schritt trifft nur noch das vordere Fahrzeug. In einem weiteren Schritt würde auch dieses Kalman-Filter nicht mehr aktiv sein.

Als Ergebnis enthält die Segmentliste S nun die Werte und die daraus resultierenden vereinfachten Konturen P ergeben sich zu:

$$S_1 = \{[\ 7\ \ 7\ ],[\ 8\ \ 6\ ],[\ 9\ \ 6\ ],[\ 10\ \ 5\ ],[\ 10\ \ 4\ ]\} \tag{15.14}$$

$$P_1 = \{[\ 7\ \ 7\ ],[\ 10\ \ 4\ ]\} \tag{15.15}$$

$$S_2 = \{[\ 11\ \ 8\ ],[\ 13\ \ 8\ ]\} \tag{15.16}$$

$$P_2 = \{[\ 11\ \ 8\ ],[\ 13\ \ 8\ ]\} \tag{15.17}$$

16 Evaluation und praktische Ergebnisse der projektspezifischen Anforderungen

Nach der Vorstellung der praktischen Umsetzung des Umfeldwahrnehmungssystems im Projekt Stadtpilot steht im Folgenden die Bewertung der Ergebnisse im Vordergrund. Basierend auf den Anforderungen aus Abschnitt 13 werden die präsentierten Algorithmen evaluiert. Hierzu werden simulierte oder reale Messdaten herangezogen. Der Evaluationsteil gliedert sich dabei in drei Abschnitte. Zunächst wird auf den Erfassungsbereich der im Stadtpilot-Projekt genutzten Sensoren eingegangen. Anschließend wird die objekthypothesen- und gitterbasierte Sensordatenfusion des Stadtpilot-Projekts detailliert behandelt. Zur Evaluation von Anforderungen werden im Folgenden zwei Methoden eingesetzt:

Auswertung von Messdaten: Die projektspezifischen Anforderungen beschreiben i. d. R. Leistungsparameter des Umfeldwahrnehmungssystems. Hierzu wurden mehrere Messfahrten durchgeführt und ausgewertet. Die Ergebnisse belegen die Erfüllung der Anforderungen in den entsprechenden Situationen.

Vergleich mit anderen Systemen und Algorithmen: Um die Vorteile einzelner Algorithmen herauszustellen, werden diese mit anderen Systemen verglichen. Einerseits kommt hier das Umfeldwahrnehmungssystem des Projekts CarOLO (Effertz, 2009) zum Einsatz und andererseits wird das konturklassifizierende Kalman-Filter mit einer traditionellen Implementierung durch ein Interacting-Multiple-Model-Filter verglichen.

Erfassungsbereiche der Sensoren

Abbildung 16.1 zeigt die Erfassungsbereiche der Sensoren am Versuchsträger des Projekts Stadtpilot. Die Messdaten wurden während verschiedener Messfahrten aufgenommen und in Fahrzeugkoordinaten dargestellt. Links ist die Übersicht über den gesamten Erfassungsbereich dargestellt. Die rechte Grafik zeigt das fahrzeugnahe Umfeld. Aus der Darstellung ergibt sich, dass ein Bereich von 360° um den Versuchsträger bis auf zwei kleine Bereiche am Heck durch die Sensorik abgedeckt wird (Anforderung SPFA23).

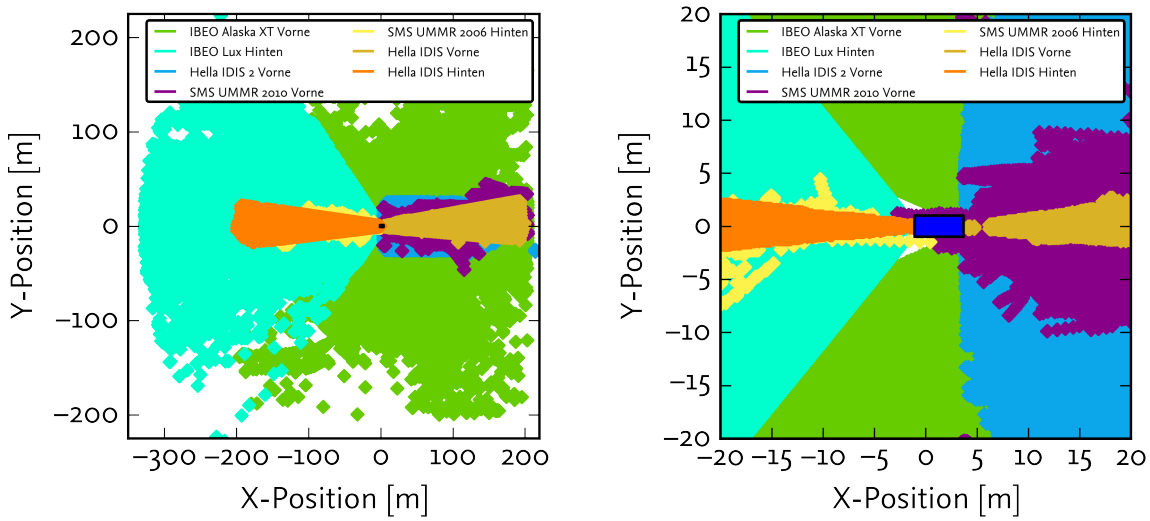


Abbildung 16.1: Reale Abdeckung der Fahrzeugumgebung durch die objekthypothesenbildenden Sensoren am Versuchsträger Leonie. Fernbereich (links), Nahbereich (rechts)

16.1 Objekthypothesenbasierte Fusion

Die Evaluation der objekthypothesenbasierten Fusion des Projekts Stadtpilot gliedert sich in drei Abschnitte: die Evaluation der projektspezifischen Anforderungen, den Vergleich des konturklassifizierenden Kalman-Filters mit einem Interacting-Multiple-Model-Filter sowie einen Vergleich zum Umfeldwahrnehmungssystem des Vorgängerprojekts CarOLO.

16.1.1 Evaluation der projektspezifischen Anforderungen

Zur Überprüfung der projektspezifischen Anforderungen werden verschiedene Messfahrten evaluiert. Jede Szene besteht aus dem Versuchsträger sowie einem anderen Fahrzeug, das mit einem hochgenauen inertial gestützten GPS-System ausgerüstet ist und als Referenz genutzt wird. Die genaue Beschreibung des Referenzsystems kann in Anhang B nachvollzogen werden.

Im Folgenden werden die Anforderungen anhand von drei Szenarien, bestehend aus dem Stadtpilot-Versuchsträger sowie dem Zielfahrzeug, untersucht:

- gleichförmige Fahrt mit abruptem Bremsmanöver
- Überholmanöver bei konstanter Geschwindigkeit
- Abbildung von Fußgängern, Radfahrern und Randbebauung auf dem Braunschweiger Stadtring

Je nach Situation werden unterschiedliche Zustandsgrößen der Sensoren oder des Tracks im Vergleich zu den Daten des Referenzfahrzeugs dargestellt. Da die Objekthypothesen der Sensoren und des Fusionsmoduls unterschiedliche geometrische Objekthypothesenmodelle aufweisen, wird für die X- und Y-Position durchgängig der Hypothesen-Schwerpunkt in den Diagrammen aufgetragen. Die Verläufe der Zustandsgrößen sind umgeben von einem halbtransparenten Bereich. Dieser zeigt die aktuelle Standardabweichung der Zustandsgröße des Sensors bzw. des Tracks auf. Im Fall der Zustandsgröße des Referenzfahrzeugs gibt dieser

Bereich die maximal zulässige Abweichung der Zustandsgröße nach Anhang C.2 an. Die Darstellung einer Kurve beginnt und endet mit einer Raute. Diese gibt den Beginn bzw. das Ende eines erfolgreichen Trackings an. Ist ein erfolgreiches Tracking kurzzeitig nicht möglich, so wird auch dies durch Rauten angezeigt. Diese Unterbrechungen zeichnen sich aber i. d. R. auch durch eine Verschiebung in der Zustandsgröße aus.

Gleichförmige Fahrt mit abruptem Bremsmanöver

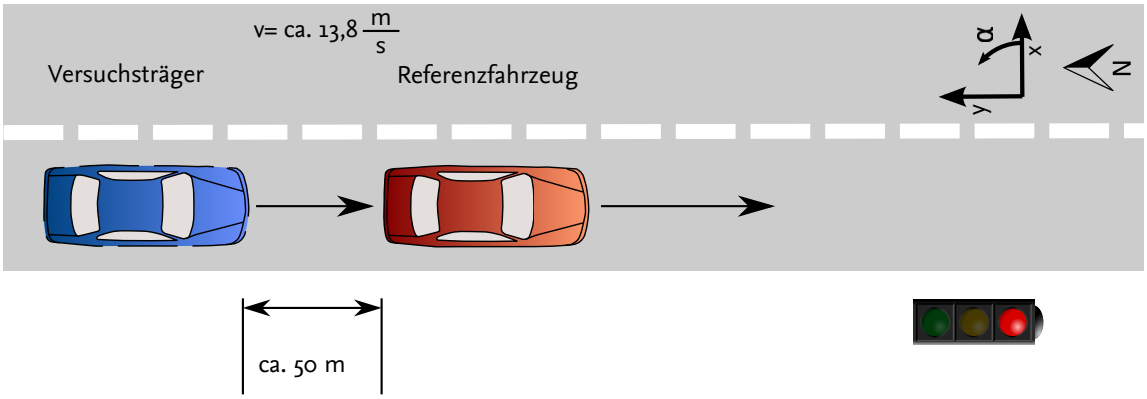


Abbildung 16.2: Situation 1: Gleichförmige Fahrt mit abruptem Bremsmanöver

Die vorliegende Szene simuliert eine übliche Folgefahrt auf dem Braunschweiger Stadtring mit kurzfristigem Farbwechsel einer Lichtsignalanlage. In der Szene bewegen sich sowohl der Versuchsträger als auch das Referenzfahrzeug mit einer Geschwindigkeit von ca. $13,8\frac{m}{s}(\approx 50\frac{km}{h}$). Nach einiger Zeit bremst das Referenzfahrzeug abrupt ab und bleibt schließlich stehen (siehe Abbildung 16.2). Die Messfahrt wurde auf dem Testgelände des Projekts Stadtpilot durchgeführt.

In den Abbildungen 16.3 bis 16.6 sind einzelne Zustandsgrößen der zum Referenzfahrzeug gehörenden Objekthypothese aufgetragen. Das Referenzfahrzeug bewegt sich während der Messfahrt im Wesentlichen entlang der Y-Achse des Weltkoordinatensystems. Insofern sind nur geringe Veränderungen in der X-Position zu beobachten (siehe Abbildung 16.3). Auffällig ist die große Unsicherheit der lateralen Position des Fahrzeugs von ca. $60cm$. Dies resultiert unmittelbar aus den verfügbaren Messdaten. So weist gerade der nach vorn gerichtete Hella IDIS-Sensor, bedingt durch die geringe Winkelauflösung, eine hohe Unsicherheit in dieser Größe auf. Zur Zuordnung eines Fahrzeugs zu einem Fahrstreifen, wie sie z. B. zur Identifikation eines Fahrzeugs zur Folgefahrt (Anforderung SPFA27) notwendig ist, sind die $60cm$ jedoch ausreichend (siehe Anhang C.2). Da sich die Fahrzeuge auf gerader Strecke entlang der Y-Achse bewegen, ist in Abbildung 16.4 eine konstante Veränderung des Wertes zu erkennen.

Abbildung 16.5 zeigt den Geschwindigkeitsverlauf über die Testfahrt. Zunächst folgt der Geschwindigkeitswert der Objekthypothese dem des Referenzfahrzeugs. Erreicht wird eine Geschwindigkeit von ca. $13,8\ \frac{m}{s}$. Zum Zeitpunkt $28,2s$ beginnt das Referenzfahrzeug abrupt abzubremsen. Dieser Geschwindigkeitsverlauf wird nicht unmittelbar nachempfunden (Latenz $\approx 1,3s$). Begründet ist diese Verzögerung in den Messdaten der Sensoren (siehe

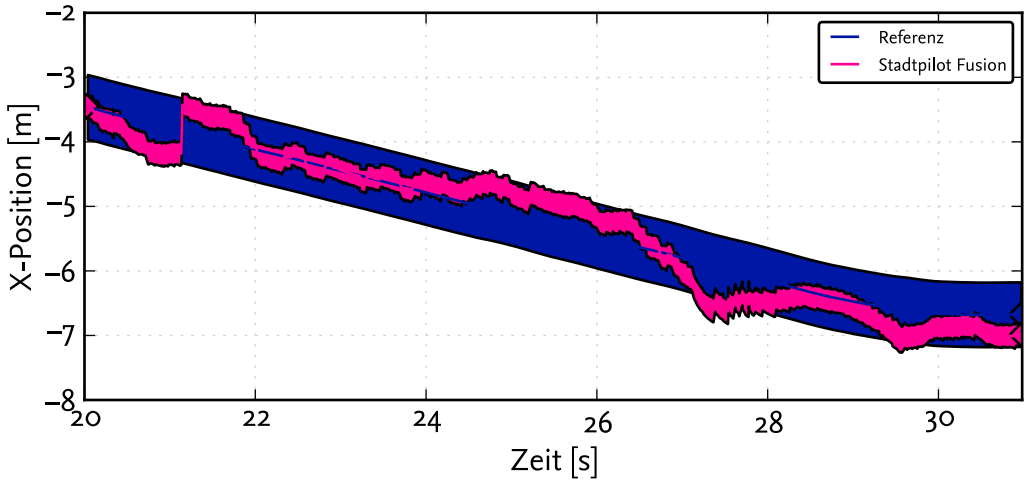


Abbildung 16.3: X-Position des Referenzfahrzeugs in Situation 1

Abbildung 16.5). Speziell der Radarsensor SMS 2010 weist in dieser Situation eine große Latenz zur Referenz auf (ca. $0,9s$ der $1,3s$ Gesamtlatenz), die, nach Ansicht des Autors, in der Ausrichtung des Sensors auf ACC-Anwendungen begründet liegt. Da dieser Sensor neben der Position auch die Geschwindigkeit direkt misst und nicht nur schätzt, wird die Standardabweichung des Geschwindigkeitswerts als entsprechend klein angenommen. Dies führt im Fehlerfall jedoch zu starken Abweichungen des Filterprozesses von der Realität. Nach einer gewissen Zeit ist die fehlerhafte Modellbeschreibung des Radarsensors jedoch so groß, dass diese Messdaten nur noch in geringem Maße Einfluss auf den Geschwindigkeitswert haben. Insgesamt sind die Abweichungen des Geschwindigkeitswerts jedoch gering und dieser weist eine ausreichende Güte für eine komfortable Folgefahrt auf (Anforderung SPFA27). Ebenfalls zeigt sich, dass von der objekthypothesenbasierten Sensordatenfusion sowohl stehende als auch sich bewegende Objekthypothesen gebildet und getrackt werden können (Anforderung SPFA24 und SPFA31).

Die letzte Grafik in Bezug auf Situation 1 zeigt die Bewegungsrichtung des Versuchsträgers (siehe Abbildung 16.6). Zunächst bewegt sich die Objekthypothese entlang der Y-Achse des Weltkoordinatensystems. Sobald das Bremsmanöver durch die Objekthypothese abgebildet wird, weicht der Wert von dem Referenzwert ab. Dies ist zunächst darin begründet, dass die Position der Objekthypothese, bedingt durch die fehlerhaften Daten des Radarsensors, von denen der anderen Sensoren abweicht. Um diese Abweichung auszugleichen, verändert sich die Geschwindigkeitsrichtung entsprechend. Sobald die Objekthypothese eine Mindestgeschwindigkeit unterschritten hat, ist eine Richtungsschätzung ohnehin nicht mehr möglich (siehe Abschnitt 14.1.3). Dies führt dazu, dass das Bewegungsmodell angepasst wird und die Bewegungsrichtung 0 angenommen wird.

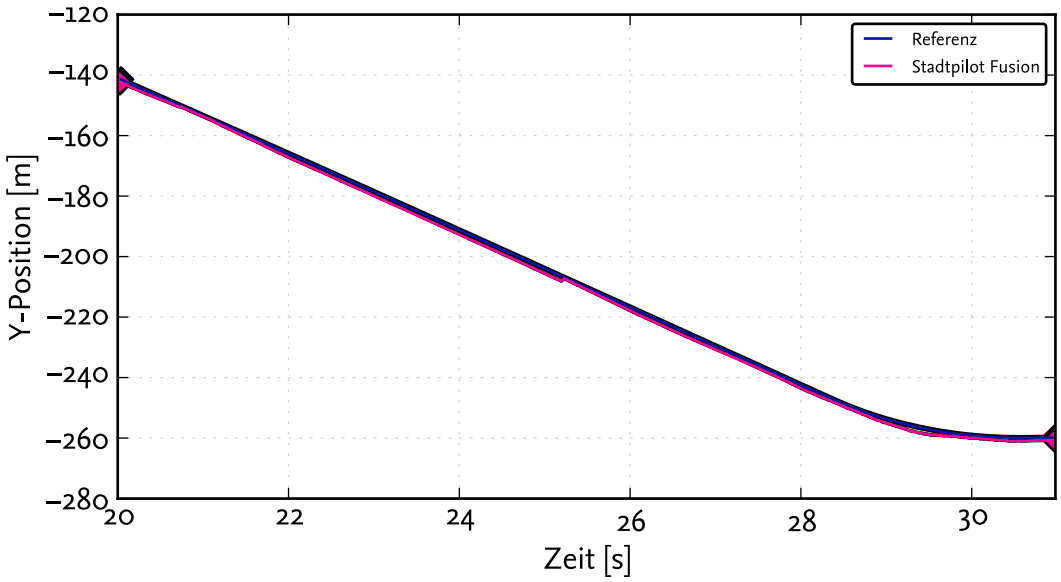


Abbildung 16.4: Y-Position des Referenzfahrzeugs in Situation 1

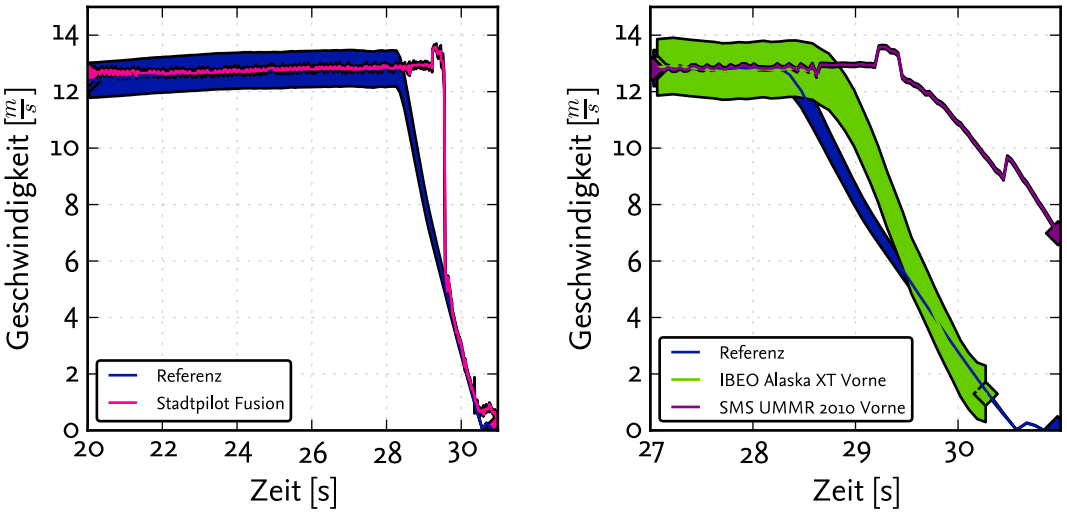


Abbildung 16.5: Geschwindigkeit des Referenzfahrzeugs in Situation 1

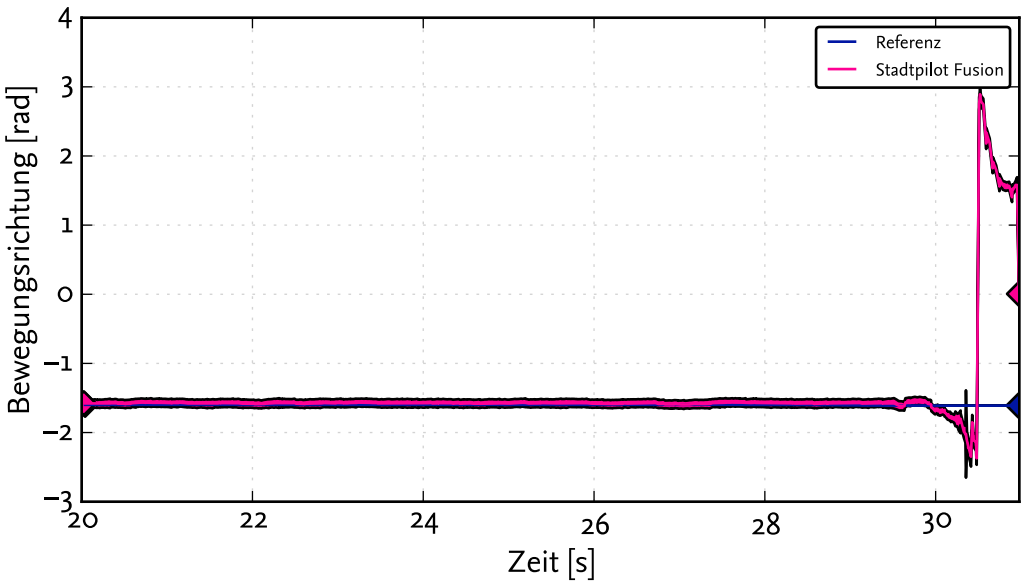


Abbildung 16.6: Bewegungsrichtung des Referenzfahrzeugs in Situation 1

Überholmanöver mit konstanter Geschwindigkeit

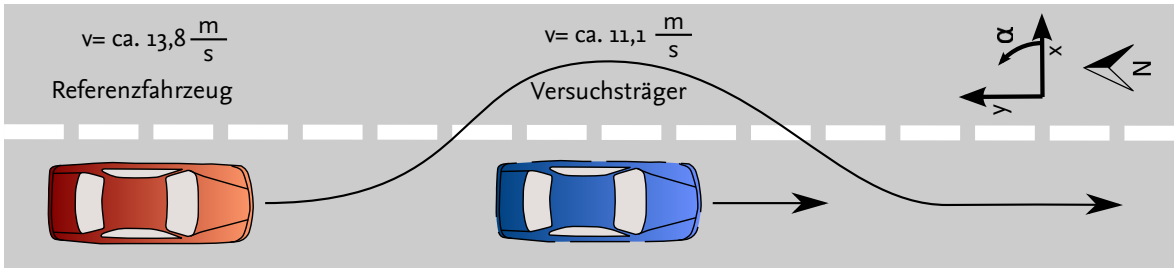


Abbildung 16.7: Situation 2: Überholmanöver mit konstanter Geschwindigkeit

Im Folgenden wird die in Abbildung 16.7 aufgezeigte Situation betrachtet. Dargestellt wird dabei ein Überholmanöver des Referenzfahrzeugs. Zunächst beschleunigen beide Fahrzeuge auf ihre Zielgeschwindigkeit. Hierbei erreicht der Versuchsträger eine geringere Geschwindigkeit (ca. $11,1\frac{m}{s} \approx 40\frac{km}{h}$) als das Referenzfahrzeug und ein Überholmanöver wird notwendig. Die Herausforderung für die Umfeldwahrnehmung liegt hier in der Verfolgung der Objekthypothese unter Berücksichtigung unterschiedlicher Betrachtungswinkel und die Übergabe eines Tracks von dem Erfassungsbereich eines Sensors in den eines Anderen.

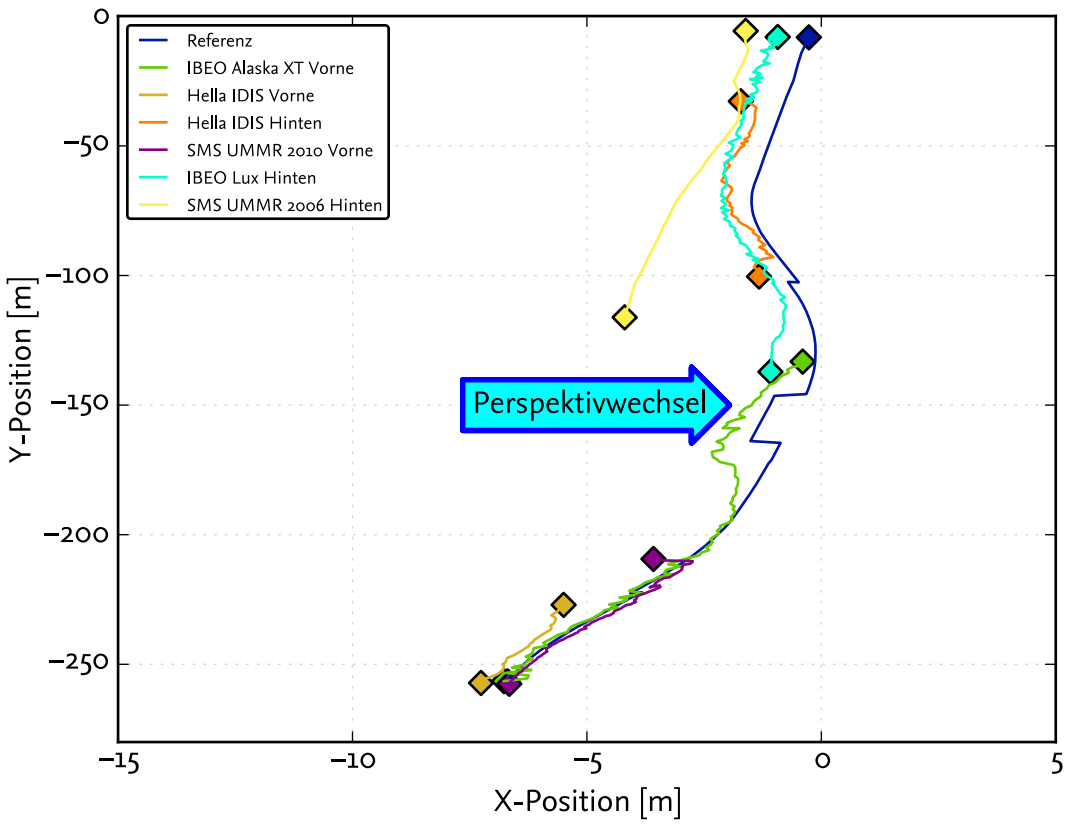


Abbildung 16.8: XY-Position der Messdaten der Sensoren

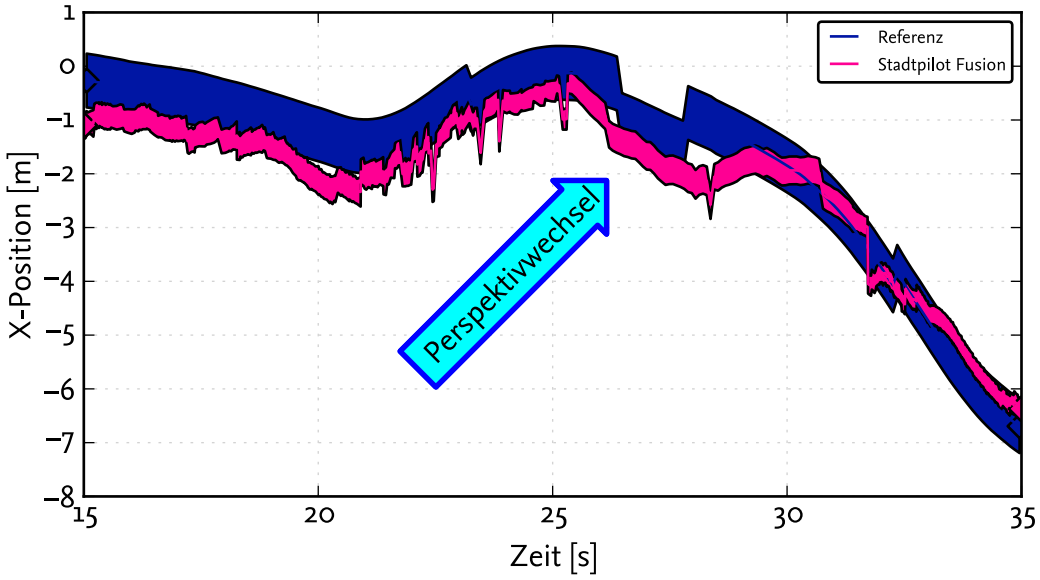


Abbildung 16.9: X-Position des Referenzfahrzeugs in Situation 2

Abbildung 16.8 zeigt die Messdaten der Sensoren während der Messfahrt. Das Referenzfahrzeug bewegt sich im Wesentlichen entlang der Y-Achse des Weltkoordinatensystems. In X-Richtung ist das Überholmanöver abgebildet. Zu beachten sind dabei die Sprünge des Schwerpunkts des Referenzwerts. Dieser kommt durch eine Konturveränderung des Referenzdatums (z. B. Übergang von einer L-Kontur hinter dem Versuchsträger zu einer Strecken-Kontur neben dem Versuchsträger) zustande und bildet sich bei den Sensoren mit Polyline-Objekthypothesenmodell ebenfalls in den Messwerten ab. Zu Beginn der Versuchsfahrt wird das Referenzfahrzeug durch die drei Sensoren am Heck des Versuchsträgers wahrgenommen. Während des Überholvorgangs können nur die beiden Laserscanner das andere Fahrzeug wahrnehmen. Nach Abschluss des Manövers nehmen die drei Sensoren im Frontbereich des Versuchsträgers das Referenzfahrzeug wahr. Auffällig ist die Objekthypothese des hinteren Radarsensors (SMS 2006 hinten, gelb), dessen Messdatum nach dem Ausscheren des Referenzfahrzeugs diesem nicht weiter folgt.

Betrachtet man nun die einzelnen Zustandsgrößen des *MainFusionModules*, so lässt sich eine Bewertung der Situation durch das Fusionsmodul durchführen. In Abbildung 16.9 ist die X-Position der Objekthypothese über der Zeit dargestellt. Zu sehen ist ein Verlauf, der dem der Referenz folgt, anfangs jedoch eine abnehmende Abweichung der beiden Schwerpunkte voneinander aufweist. Bei ca. $25s$ findet die Übergabe der Objekthypothese zwischen den Sensoren statt. In der Zeit vor $25s$ wird die Objekthypothese durch den IBEO Lux-Sensor gestützt, während diese anschließend in den Erfassungsbereich des IBEO Alaska XT-Sensors eintritt und durch dessen Messdaten gestützt wird. Während der Perspektivwechsel überschreitet die Abweichung den maximal zulässigen Wert kurzzeitig. Insgesamt lässt sich jedoch sagen, dass die Abweichung des Schwerpunkts während der Messfahrt im Wesentlichen unter den in Anhang C.2 geforderten $\pm 50cm$ bleibt (Anforderung SPFA28).

Das Referenzfahrzeug beschleunigt zunächst auf die Zielgeschwindigkeit von ca. $13,8\frac{m}{s}$ (Abbildung 16.10). Die zugehörige Zustandsgröße der Objekthypothese bildet diesen Wert

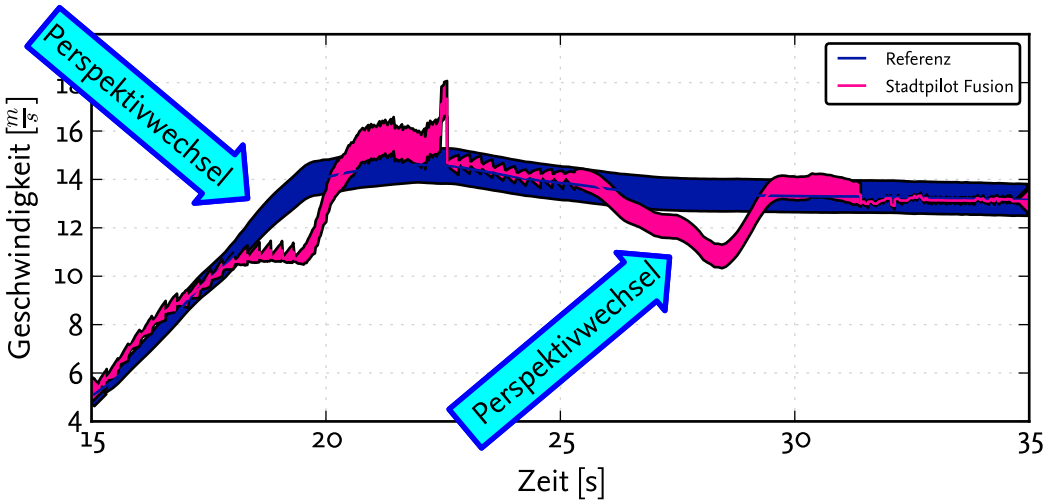


Abbildung 16.10: Geschwindigkeit des Referenzfahrzeugs in Situation 2

bis auf die Zeit während der Perspektivwechsel ausreichend gut ab. Gefordert war hier eine Genauigkeit von 5%. Diese wird beim ersten und dritten Perspektivwechsel kurzzeitig unterschritten, folgt anschließend jedoch rasch wieder dem Referenzwert (Anforderung SPFA27).

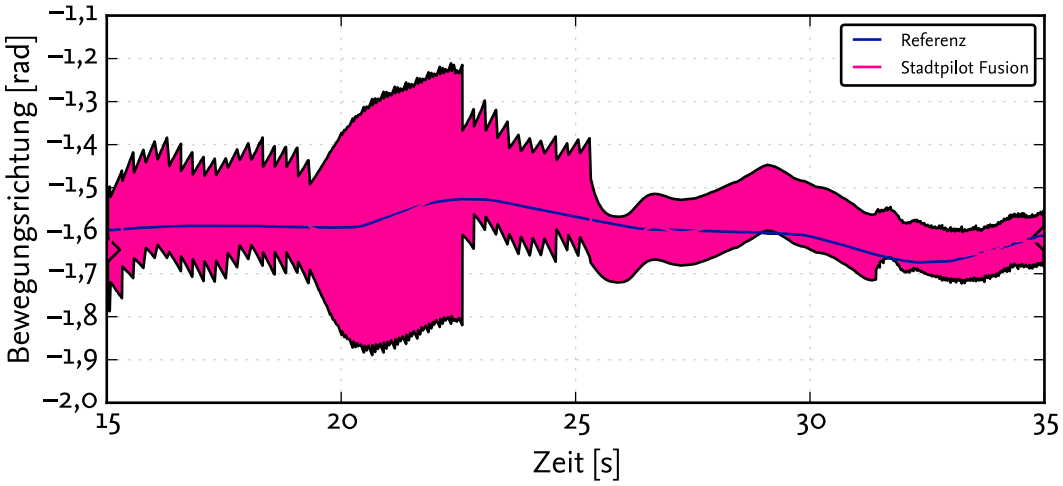


Abbildung 16.11: Bewegungsrichtung des Referenzfahrzeugs in Situation 2

Die Bewegungsrichtung der Objekthypothese wird zur Zielauswahl und zur Entscheidung darüber benötigt, ob ein Fahrzeug einen möglichen Fahrstreifenwechsel des Versuchsträgers behindert. In Abbildung 16.11 wird die aktuelle Bewegungsrichtung des Referenzfahrzeugs dargestellt. Zu sehen ist, dass diese der Referenz folgt. Bei ca. 25s erfolgt die Übergabe zwischen den beiden Sensoren. Hier ist kurzzeitig eine starke Abweichung zu sehen, die jedoch innerhalb sehr kurzer Zeit ausgeglichen wird.

Daten während Messfahrten auf dem Braunschweiger Stadtring

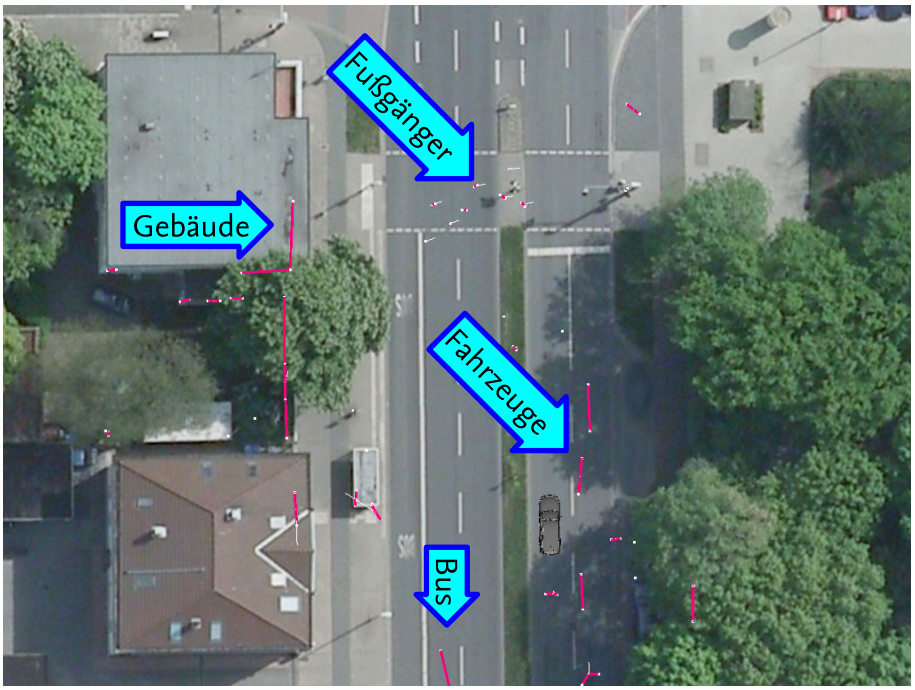


Abbildung 16.12: Messdaten von einer Messfahrt auf dem Braunschweiger Stadtring (Kartengrundlage: ©Stadt Braunschweig, Abteilung Geoinformationen (Nr. 011/2010)

Abbildung 16.12 zeigt Messdaten einer Messfahrt auf dem Braunschweiger Stadtring über einer statischen Luftaufnahme. Der Versuchsträger ist rechts unten als 3D-Modell dargestellt. Umgeben wird er von einer Reihe von anderen Fahrzeugen, deren Position und Ausmaße wahrgenommen werden. Durch die Verdeckung durch andere Verkehrsteilnehmer lässt sich die Randbebauung aufgrund der optischen Sensoren nur bedingt wahrnehmen. Auf der anderen Straßenseite ist jedoch das Gebäude sowie der zugehörige Zaun durch Objekthypothesen abgebildet (Anforderung SPFA30). Im oberen Teil der Darstellung sind eine Reihe von Fußgängern abgebildet, die aus einem Bus ausgestiegen sind und aktuell die Straße kreuzen. Sie sind durch einzelne Objekthypothesen abgebildet (Anforderung SPFA25).

Nicht nur Fußgänger werden durch die Objekthypothesen des *MainFusionModules* abgebildet. Auch einzelne Fahrradfahrer werden korrekt wahrgenommen, solange sie nicht durch Randbebauung verdeckt sind. In Abbildung 16.13 sind drei Radfahrer zu erkennen, die sich auf dem Radweg fortbewegen (Anforderung SPFA25).

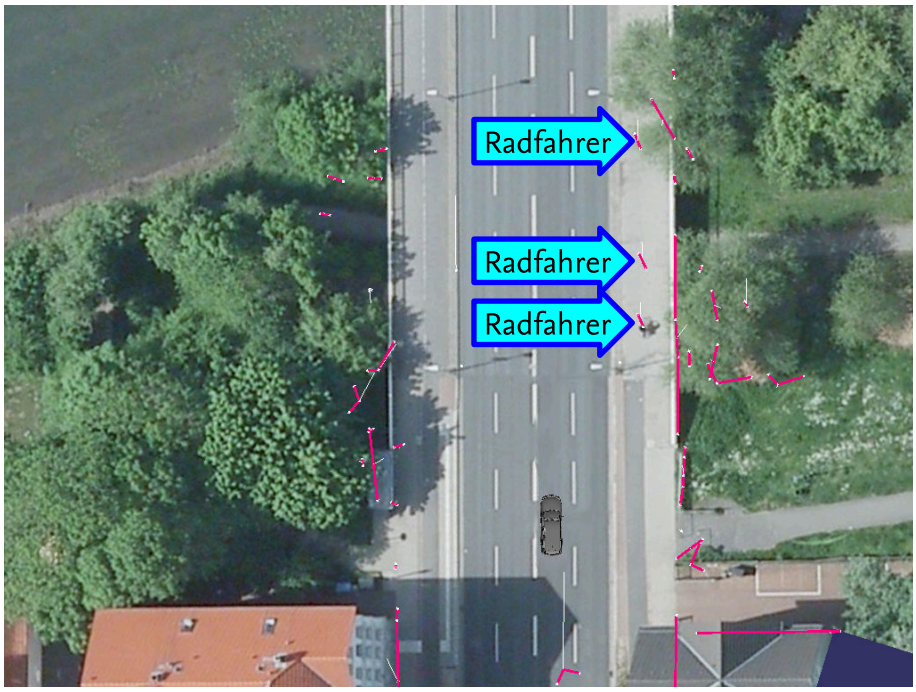


Abbildung 16.13: Messdaten von einer Messfahrt auf dem Braunschweiger Stadtring (Kartengrundlage: ©Stadt Braunschweig, Abteilung Geoinformationen (Nr. 011/2010)

Unterdrückung von ungewollten Objekthypothesen

In Abschnitt 14.3 wird ein Algorithmus zur Reduktion von Sensorgeistern beschrieben. Hierbei werden Schwächen einzelner Sensoren durch die Fähigkeiten anderer ausgeglichen. Genutzt werden dazu vor allem Redundanzeigenschaften der Erfassungsbereiche der Sensorik. Sollte ein Objekt aufgrund von überlagerten Erfassungsbereichen eigentlich durch mehr als einen Sensor wahrgenommen werden, wird aber aktuell nur von einem Sensor wahrgenommen, so wird davon ausgegangen, dass diese Konfliktsituation einen Messfehler darstellt. Zur Auflösung des Konflikts wird der Sensorgeist (siehe Definition auf Seite 121) unterdrückt, um die Fahraufgabe nicht zu beeinträchtigen (Anforderung SPFA32). Die Wahrscheinlichkeit für einen Messfehler ist um so größer, je mehr Sensoren ein Objekt nicht wahrnehmen. Im Front- und Heckbereich des Fahrzeugs wird deshalb eine dreifache Redundanz, z. T. mit unterschiedlichen Messprinzipien, zur Reduktion von Sensorgeistern genutzt.

Die Auswirkungen der Unterdrückung von Objekthypothesen lassen sich anhand deren Häufigkeitsverteilung einer Messfahrt auf dem Braunschweiger Stadtring (Dauer ca. 15 min) beurteilen (siehe Abbildung 16.14). Die meisten Sensorgeister sind in drei Ursachen begründet. Zunächst werden sporadische Messfehler unterdrückt. Diese Messfehler sind durch das Grundrauschen entlang der X-Achse sichtbar. Die Breite dieses Streifens resultiert vor allem aus der Breite der Fahrbahn während der Testfahrt, aber auch aus der Ausrichtung der Sensorik. Die zweite Gruppe von unterdrückten Objekthypothesen wird durch Messfehler der Sensoren hervorgerufen. So erzeugt der vordere Radarsensor durchgängig zwei Objekthypothesen links und rechts des Fahrzeugs, welche durch Reflexionen am Versuchsträger bedingt sind. Da diese nicht durch andere Sensoren gestützt werden, werden sie

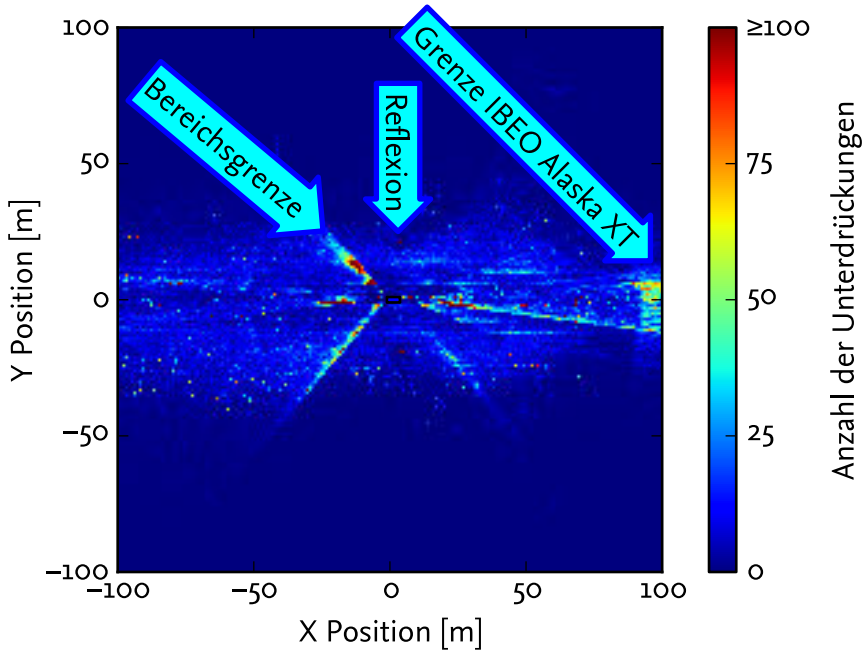


Abbildung 16.14: Qualitative Häufigkeitsverteilung der durch die Komponente *NewTrackSelektion* unterdrückten Objekthypothesen

unterdrückt. Als dritte Gruppe werden Messungen unterdrückt, die aus einer Abweichung der Sensordokumentation mit der Realität resultieren. So misst der hintere Laserscanner immer wieder Objekthypothesen außerhalb seines dokumentierten Erfassungsbereichs. Ein ähnliches Phänomen ergibt sich in der Kombination der vorderen Laserscanner mit dem Multi-Beam-Lidarsensor. Bei der Konfiguration wird davon ausgegangen, dass der IBEO Alaska XT die Objekthypothesen des Hella IDIS bestätigen kann. Da der IBEO Alaska XT aber den Erfassungsbereich bis $100m$, bedingt durch eine entfernungsabhängige Priorisierung von Objekthypothesen (Parametrierung des Sensors), in der Realität nicht immer wahrnimmt, werden diese unterdrückt, bis sie sich auf ca. $80m$ dem Versuchsträger genähert haben.

Latenz des *MainFusionModules*

Die vorangegangene Untersuchung über die Latenz der Softwarearchitektur lässt sich auch für das *MainFusionModule* durchführen (Anforderung SPNF33). In Abbildung 16.15 ist die Latenz einer Objekthypothese bei unterschiedlicher Anzahl an Tracks in der *TrackDatabase* aufgetragen. Die Latenz steigt polynomiell mit der Anzahl der Tracks in der *TrackDatabase*. Dies resultiert vor allem aus dem eingesetzten Zuordnungsalgorithmus, welcher ein quadratisches Laufzeitverhalten aufweist. Während der Testfahrten auf dem Braunschweiger Stadtring treten i. d. R. ca. 100 Tracks gleichzeitig auf. Die Latenz von $32,2ms$ (Mittelwert) (95%-Intervall: $29,9ms$ - $35,7ms$)[1] liegt innerhalb des Zielbereichs (siehe Anhang C).

[1]Intel i7 2640M, 2,8 GHz, 4 GB Speicher, Linux 3.2.0, ADTF 2.7.0

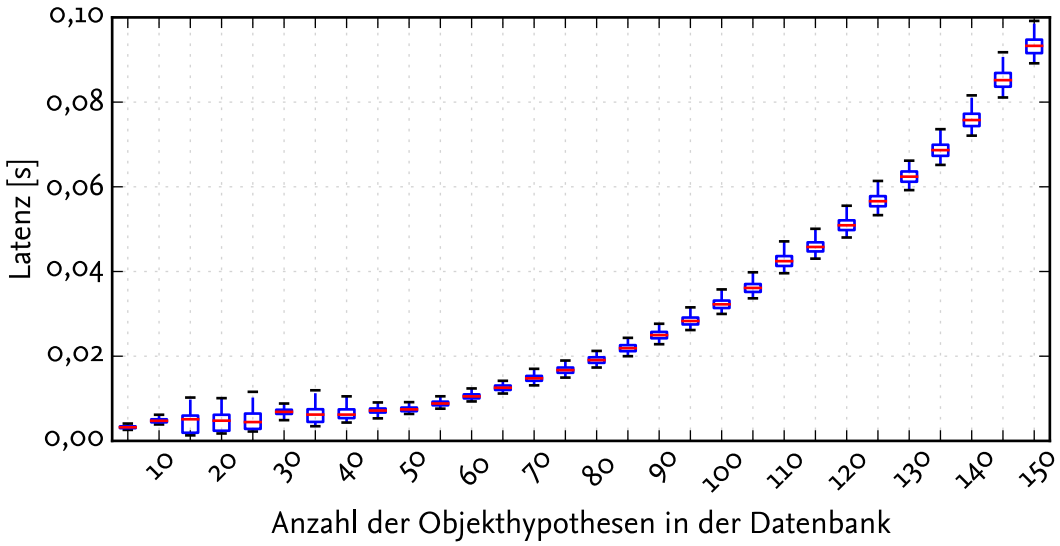


Abbildung 16.15: Latenz bei der Verarbeitung von Objekthypothesen im *MainFusionModule* (Box: mittleres Quartil, Whisker: 95%-Intervall)

Weitere Anforderungen

Mithilfe der Komponente *TrackManagement* wird die Anforderung SPNF34 erfüllt. Neben der Entfernung von Objekthypothesen aus der *TrackDatabase*, die über einen längeren Zeitraum keine Aktualisierung erfahren haben, wird der Datenbestand nach unrealistischen Daten durchsucht. Werden Objekthypothesen mit diesen Eigenschaften gefunden, so werden die Daten ggf. korrigiert oder die zugehörige Objekthypothese entfernt.

Innerhalb der *Filter*-Komponente wurden filterspezifische Überprüfungen der vorhandenen Daten umgesetzt. Hierbei wird das Geometrie- und Bewegungsmodell jeweils getrennt untersucht. Für das Geometriemodell werden zwei Untersuchungen durchgeführt: Zunächst werden die Zustandsgrößen auf nicht numerische Werte (z. B. $\pm\infty, NaN$) untersucht. Darüber hinaus wird die Kontur auf Punkte untersucht, welche sehr nah beieinanderliegen. Diese Punkte werden entfernt. Für das Bewegungsmodell wird nur die Überprüfung auf gültige numerische Werte durchgeführt.

Übersicht über die mithilfe der objekthypothesenbasierten Sensordatenfusion erfüllten Anforderungen

Die vorangegangenen Absätze lassen sich wie folgt zusammenfassen:

Anforderung	Erfüllt		Anforderung	Erfüllt
SPFA23	✓		SPFA30	✓(soweit durch Sensoren erfasst)
SPFA24	✓		SPFA31	✓
SPFA25	✓		SPFA32	✓
SPFA27	✓		SPNF33	✓
SPFA28	✓		SPNF34	✓

16.1.2 Vergleich von Konturen und Berechnung des MSE

In den bisherigen Diagrammen wurden stets einzelne Zustandsgrößen einer Objekthypothesenkontur genutzt, um zwei Objekthypothesen miteinander zu vergleichen. Sollen jedoch mehrere normalverteilte Zustandsgrößen gleichzeitig Eingang in einen Vergleichswert finden, so kann der Mean Squared Error (MSE) genutzt werden. Er stellt eine einfache Form der Repräsentation einer Abweichung vom Sollwert dar. Der MSE ist definiert als Quadrat der Abweichung vom Sollwert (Devore u. Berk, 2012, Seite 334ff):

$$E(\hat{x}) \quad = \quad (\hat{x} - x)^2 \tag{16.1}$$

Wird diese Formel nun auf den Vergleich zweier Objekthypothesen, wie sie von dem *MainFusionModule* erzeugt werden, angewendet, so stellt sich die Frage, wie mit unterschiedlichen Ausprägungen von $\hat{x}$ und x umzugehen ist. Da sich eine Objekthypothese in zwei unterschiedliche Modelle aufteilt, wird der MSE für diese Teile getrennt betrachtet.

Der MSE des Bewegungsmodells der Objekthypothese und einer Referenz lässt sich direkt aus Formel 16.1 berechnen, da der Zustandsvektor der Referenzobjekthypothese dem der aktuellen Objekthypothese entspricht:

$$E(\text{Bewegungsmodell}) \quad = \quad \left(\widehat{\begin{bmatrix} v \\ \alpha \\ \dot{v} \\ \dot{\alpha} \end{bmatrix}} - \begin{bmatrix} v \\ \alpha \\ \dot{v} \\ \dot{\alpha} \end{bmatrix} \right)^2 \tag{16.2}$$

Zur Berechnung des MSE für das geometrische Modell $E(\text{Geometriemodell})$ ist ein algorithmisches Verfahren notwendig, da die Zustandsvektoren des Referenzdatums und der Objekthypothese nicht die gleiche Anzahl an Elementen aufweisen müssen. Um dieser Herausforderung zu begegnen, werden die ersten beiden Phasen des in Abschnitt 14.1.2 beschriebenen Konturpunktverarbeitungsalgorithmus genutzt. Mit ihnen lässt sich die Differenz der aktuellen Kontur mit der Referenzkontur bestimmen. Stützpunkte, die nicht zugeordnet werden können, fließen nicht in die Berechnung des MSE ein.

Der Gesamt-MSE einer Objekthypothese $E(\text{Objekthypothese})$ bestimmt sich schließlich aus der Addition der MSE der einzelnen Zustandsgrößen:

$$E(\text{Objekthypothese}) \quad = \quad E(\text{Bewegungsmodell}) + E(\text{Geometriemodell}) \tag{16.3}$$

16.2 Vergleich des konturklassifizierenden Kalman-Filters mit einem IMM-Filter

Das IMM-Filter enthält folgende Modelle:

- Punkt-Geometriemodell mit CV-Bewegungsmodell
- Strecken-Geometriemodell mit CV-Bewegungsmodell

Das im *MainFusionModule* umgesetzte konturschätzende Kalman-Filter beschreibt eine Objekthypothese mit an das Verhalten des wahrgenommenen Objekts angepassten Modellen. Würde das Filter ausschließlich unterschiedliche Bewegungsmodelle einsetzen, wäre die Nutzung eines Interacting-Multiple-Model-Filters (IMM) üblich (Blom, 1984). Daraus ergibt sich die Frage, ob sich dieser Filteralgorithmus auch für die Schätzung des Geometriemodells eignet. Aus diesem Grund wird im Folgenden das konturschätzende Kalman-Filter mit einem IMM-Filter verglichen. Da IMM-Filter vor allem bei einer kleinen Anzahl von möglichst unterschiedlichen Modellen gute Ergebnisse liefern (Li u. Bar-Shalom, 1992), wird das Vergleichsfilter nur zwei unterschiedliche Modelle enthalten. Darüber hinaus nutzen die Modelle das gleiche Bewegungsmodell.

Die Messwertverarbeitung des IMM-Filters geschieht analog zur Verarbeitung des konturschätzenden Kalman-Filters aus Abschnitt 14.1. Dabei wird zunächst eine Konturpunktverarbeitung durchgeführt und die anschließende durchschnittliche Abweichung des prädizierten Tracks vom Messwert als Eingangsgröße des Filters genutzt. Die Aktualisierung des Filters geschieht entsprechend dem Verfahren aus Abschnitt 2.3.2. Anhand des Vektors μ wird nach dem Aktualisierungsschritt eines der beiden Modelle ausgewählt.

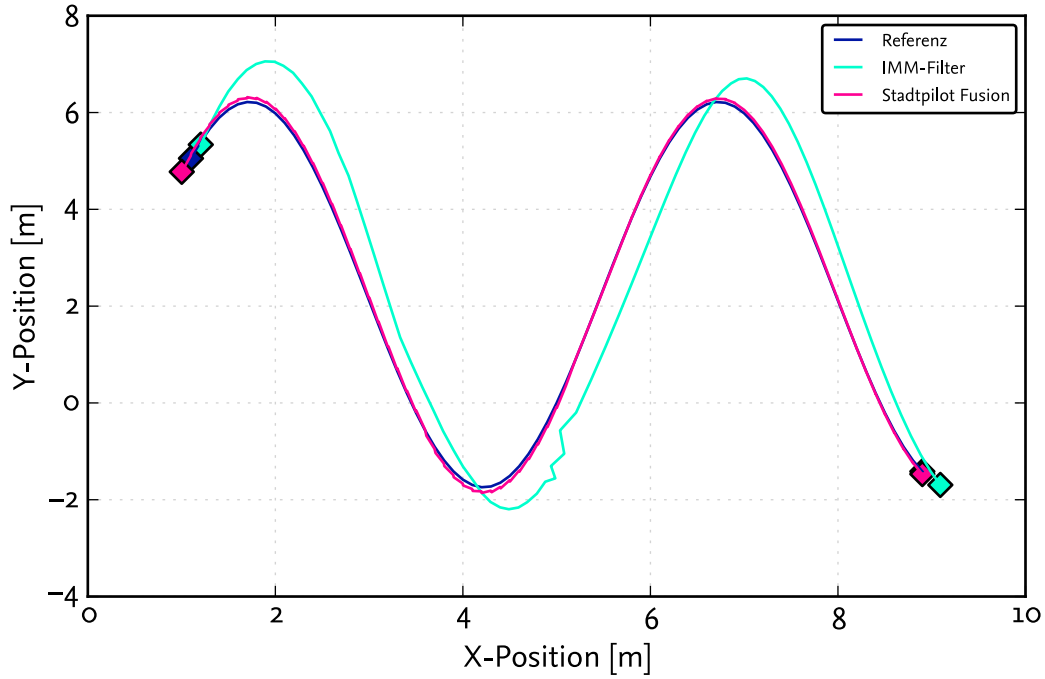


Abbildung 16.16: XY-Position des simulierten Objekts

Für den Vergleich bewegt sich das Objekt auf einer simulierten sinusförmigen Trajektorie. Dabei wird es von zwei unterschiedlichen rauschfreien Sensoren wahrgenommen. Der eine Sensor repräsentiert Objekthypothesen durch ein Punkt-Geometriemodell, der andere durch das Strecken-Geometriemodell. Das Objekt führt ca. eineinhalb Sinusschwingungen durch. Während der ersten Schwingung wird das Objekt von beiden Sensoren als Punkt (bzw. als Strecke mit sehr kurzer Länge) wahrgenommen. Anschließend repräsentiert der Streckensensor das Objekt mit einer Breite von $2m$. Die Matrizen des Zustandsschätzers und der

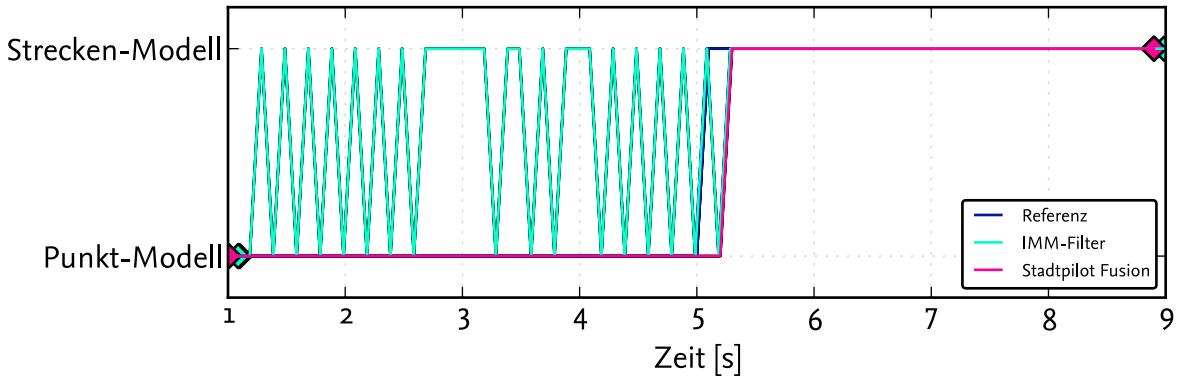


Abbildung 16.17: Aktives Geometriemodell des simulierten Objekts

Modellwahrscheinlichkeit werden zu Beginn mit korrekten Werten vorinitialisiert.

Abbildung 16.16 zeigt die Trajektorie des simulierten Objekts, wie es von den beiden Filtern verfolgt wird. Zum Zeitpunkt $5s$ schaltet das Modell des Referenzobjekts von Punkt- auf das Strecken-Modell um. An dieser Stelle ist kurzzeitig eine deutliche Verschlechterung der Position in der Schätzung des IMM-Filters zu sehen. Darüber hinaus folgt das IMM-Filter der Referenztrajektorie weniger exakt als das konturschätzende Kalman-Filter.

Betrachtet man das aktive Modell der Filter, so lässt sich erkennen, dass das IMM-Filter die beiden Geometriemodelle zunächst nicht differenzieren kann (siehe Abbildung 16.17). Das konturschätzende Kalman-Filter kann dies durch seine evidenzbasierte Modellschätzung jedoch leisten und folgt der Referenz mit einem Takt Verzögerung. Der Grund für diese „Unentschiedenheit“ liegt in der Ähnlichkeit einer kurzen Strecke und eines Punktes. Das konturschätzende Kalman-Filter kann aufgrund der Zuordnung der Objekthypothesen zu Metakonturen diese Korrespondenz erkennen und sich für das richtige Modell entscheiden.

Betrachtet man nun die gleiche Trajektorie und addiert ein gaußsches weißes Rauschen mit einer Standardabweichung von $0,1m$ auf die Messwerte, so ergibt sich ein anderes Bild (siehe Abbildung 16.18). Die Ergebnisse des konturschätzenden Kalman-Filters haben sich minimal verschlechtert. Bei den Werten des IMM-Filters ist die Degradation jedoch erheblich größer. Sie folgen, gerade in der zweiten Hälfte, nur noch grob den Werten der Referenz.

Diese Verschlechterung der Werte lässt sich vor allem mit der Unsicherheit bei der Modellauswahl erklären (siehe Abbildung 16.19). War im rauschfreien Fall noch eine Unentschiedenheit in der ersten Hälfte der Simulation zu sehen, so ist diese Unsicherheit nun auch in der zweiten Hälfte sichtbar. Erst als sich das IMM-Filter auf ein Modell festlegt, nimmt die Abweichung des Filterergebnisses von denen der Referenz ab. Auch in der ersten Hälfte legt sich das Filter nicht eindeutig auf ein Modell fest, der Unterschied zwischen einer sehr kurzen Strecke und einem Punkt ist jedoch so gering, dass dies im Ergebnis zu keinen größeren Abweichungen führt.

Ein Blick auf den Mean Squared Error der beiden Filter zeigt deutlich den Unterschied zwischen den Filterergebnissen. Während zwischen den MSE-Werten des konturklassifizierenden Kalman-Filters kein Unterschied innerhalb der ersten 4 Nachkommastellen besteht, liegen die Werte des IMM-Filters erheblich höher (siehe Tabelle 16.1).

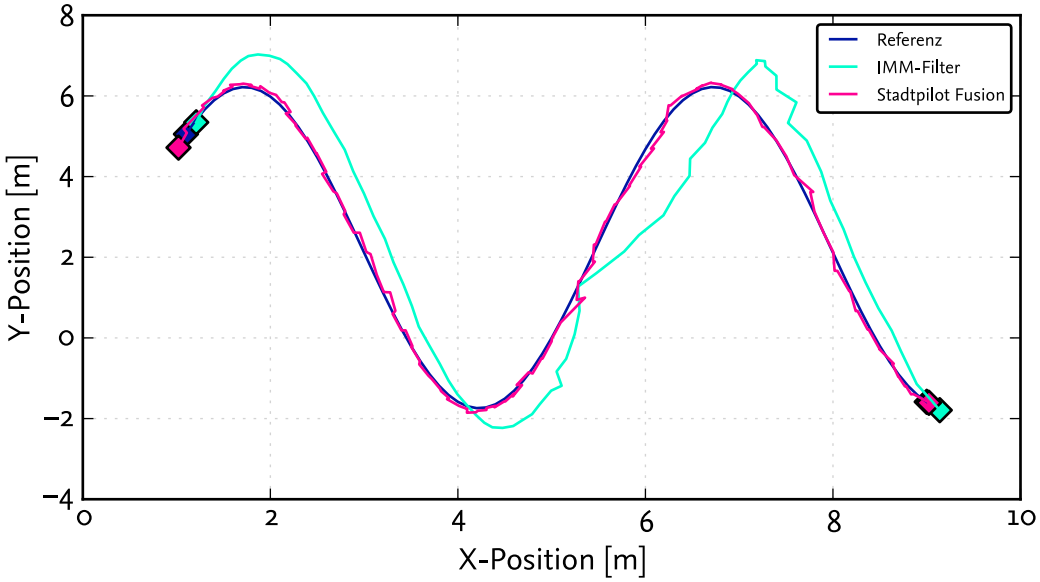


Abbildung 16.18: XY-Position des simulierten Objekts mit einem gaußschen weißen Rauschen von $0,1m$

Filter	Rauschen	MSE
IMM-Filter	$0m$	$0,2649m^2$
IMM-Filter	$0,1m$	$0,4477m^2$
Konturklassifizierender Kalman-Filter	$0m$	$0,0027m^2$
Konturklassifizierender Kalman-Filter	$0,1m$	$0,0027m^2$

Tabelle 16.1: Vergleich des konturklassifizierenden Kalman-Filters mit einem IMM-Filter anhand des MSE

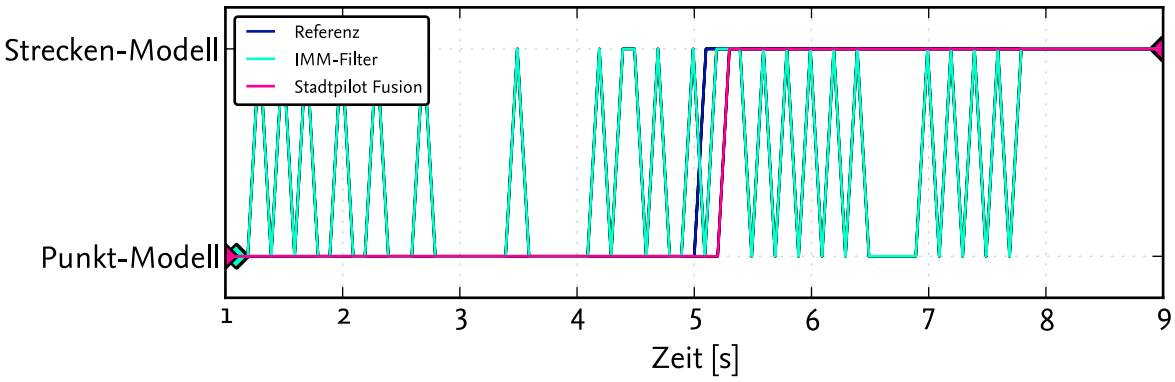


Abbildung 16.19: Aktives Geometriemodell des simulierten Objekts mit einem gaußschen weißen Rauschen von $0,1m$

16.3 Vergleich mit der Fusion des Projekts CarOLO

Die objekthypothesenbasierte Fusion des Projekts Stadtpilot basiert auf den Erfahrungen des Projekts CarOLO während der DARPA Urban Challenge (siehe Abschnitt 3). Die im Evaluationsteil der zugehörigen Dissertation beschriebenen Situationen werden deshalb im Folgenden im Vergleich betrachtet (Effertz, 2009, Seite 134ff).

Für das Vorgängersystem wurden drei Situationen evaluiert:

- geradlinige Fahrt mit konstanter Geschwindigkeit
- geradlinige Fahrt mit konstanter Beschleunigung
- Fahrt in Schlangenlinien

Diese Situationen wurden mit dem Versuchsträger des Projekts Stadtpilot entsprechend der Situationsbeschreibung aus Effertz (2009, Seite 134ff) nachgestellt. Als Zielfahrzeug wurde das Referenzfahrzeug aus Anhang B genutzt. Da der Versuchsträger des Projekts Stadtpilot und das Fahrzeug des Teams CarOLO im Frontbereich die gleichen Sensoren für die objekthypothesenbasierte Sensordatenfusion nutzen bzw. genutzt haben, werden für den Vergleich dieselben Messdaten mit dem originalen Softwaremodul des Teams CarOLO und dem *MainFusionModule* des Projekts Stadtpilot verarbeitet. Als Qualitätskriterien für den Vergleich werden die Abweichung der Zustandsgrößen von der Referenz und der Verlust des Trackings zugrunde gelegt. Für die Beschleunigungswerte steht kein Referenzwert zur Verfügung.

Geradlinige Fahrt mit konstanter Geschwindigkeit

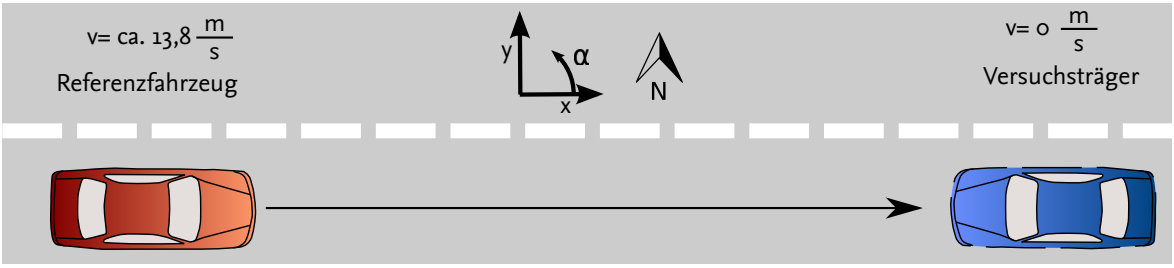


Abbildung 16.20: Situation 1 nach Effertz (2009, Seite 135f): Geradlinige Fahrt mit konstanter Geschwindigkeit

Im Versuch befindet sich der Versuchsträger in Ruhe, während das Zielfahrzeug sich auf diesen mit ca. $13,8\frac{m}{s}$zubewegt (siehe Abbildung 16.20). Dabei bewegt sich das Ziel entlang der X-Achse des Weltkoordinatensystems.

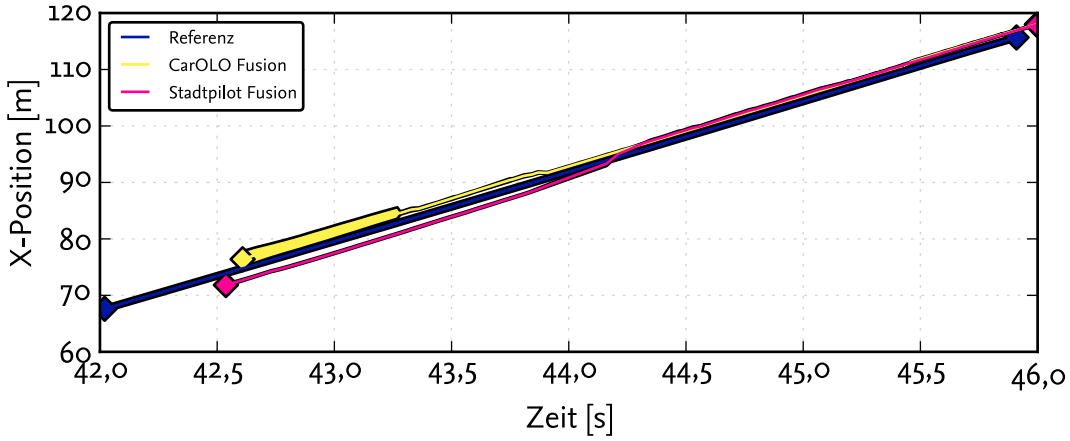


Abbildung 16.21: X-Position des Referenzfahrzeugs in Situation 1 nach Effertz (2009, Seite 135)

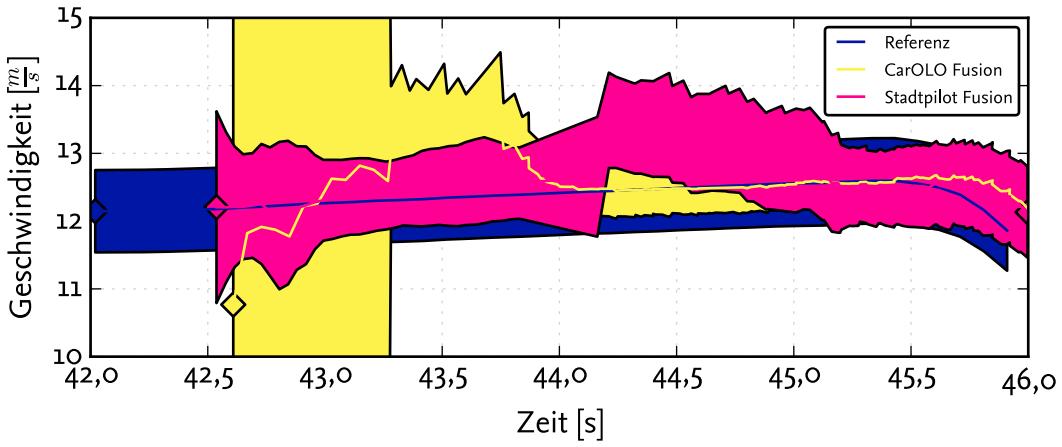


Abbildung 16.22: Geschwindigkeit des Referenzfahrzeugs in Situation 1 nach Effertz (2009, Seite 135)

Die X-Position der zum Referenzfahrzeug gehörenden Objekthypothese folgt wie erwartet mit geringen Abweichungen der Position des Referenzfahrzeugs (siehe Abbildung 16.21). Ein qualitativer Unterschied zwischen der Sensordatenfusion des CarOLO Projekts und des Projekts Stadtpilot ist hier nicht erkennbar. Bei der Betrachtung der Geschwindigkeits- und Richtungsinformation folgen ebenfalls beide Fusionen den Werten der Referenz (siehe Abbildungen 16.22 und 16.23). Auffällig ist jedoch die große Standardabweichung der Fusion des CarOLO Projekts. Diese wird erst effektiv kleiner, als das Referenzfahrzeug nah genug ist und ein dritter Sensor mit in die Fusion einbezogen wird. In Abbildung 16.24 ist die Beschleunigung der Objekthypothese dargestellt. Der Referenzwert steht dabei leider nicht zur Verfügung (siehe Anhang B). Dabei ist zu Beginn des Verlaufs der CarOLO-Objekthypothese eine kurzeitige erhebliche Abweichung zu sehen.

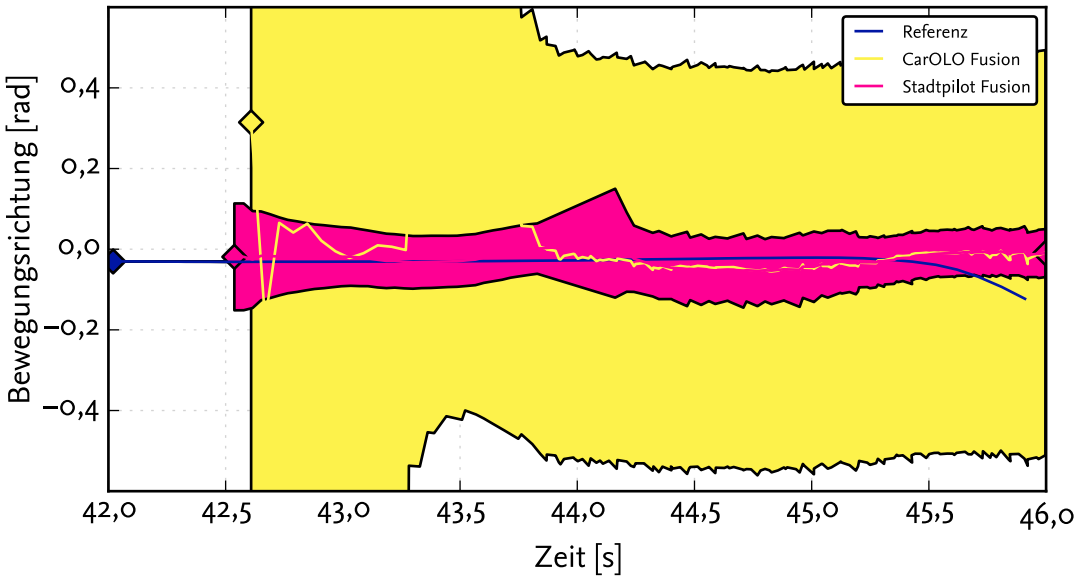


Abbildung 16.23: Bewegungsrichtung des Referenzfahrzeugs in Situation 1 nach Effertz (2009, Seite 135)

Geradlinige Fahrt mit konstanter Beschleunigung

Die zweite Situation zeigt eine geradlinige Fahrt des Referenzfahrzeugs auf den Versuchsträger zu. Dabei erfährt das Referenzfahrzeug eine konstante Beschleunigung mit anschließendem scharfen Bremsmanöver (siehe Abbildung 16.25). Das Ziel bewegt sich dabei erneut entlang der X-Achse des Weltkoordinatensystems.

Auch in dieser Situation folgen die Zustandsgrößen der Objekthypothesen beider Fusionen denen des Referenzfahrzeugs (siehe Abbildungen 16.26 bis 16.29). Die Fusion des Projekts CarOLO kann dem Track jedoch bei ca. $45,5s$ für kurze Zeit nicht länger folgen und setzt hier einen neuen Track auf.

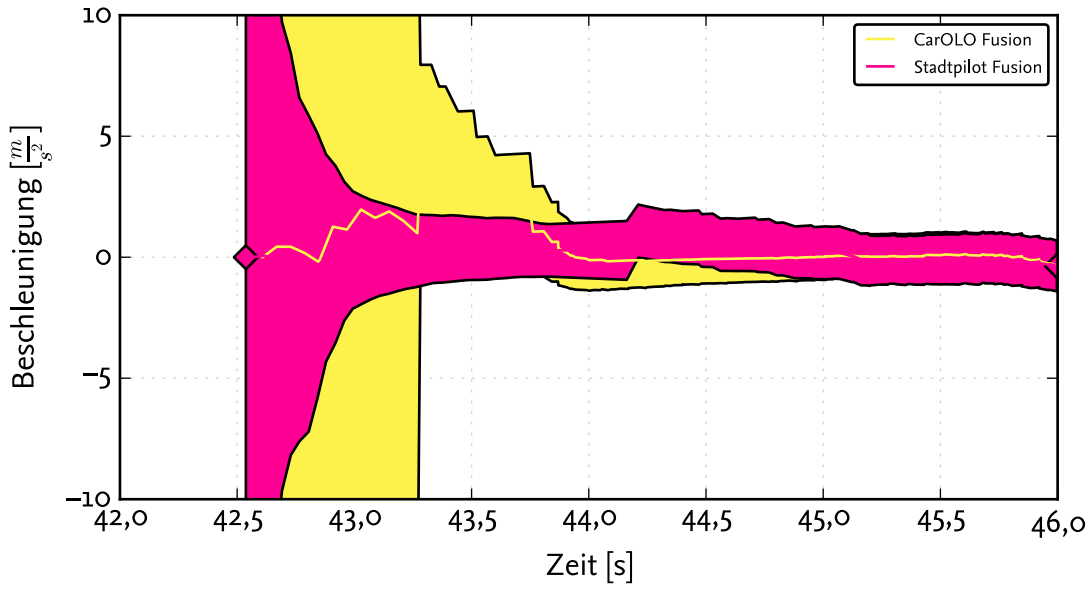


Abbildung 16.24: Beschleunigung des Referenzfahrzeugs in Situation 1 nach Effertz (2009, Seite 135)

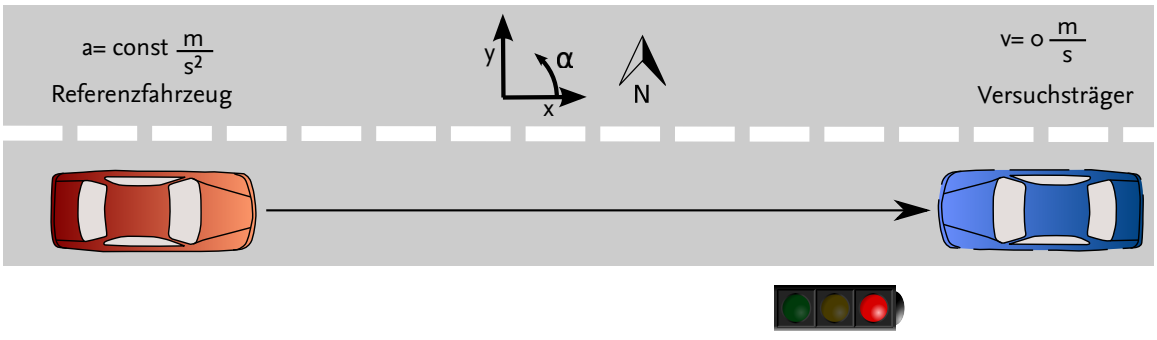


Abbildung 16.25: Situation 2 nach Effertz (2009, Seite 137f): Geradlinige Fahrt mit konstanter Beschleunigung

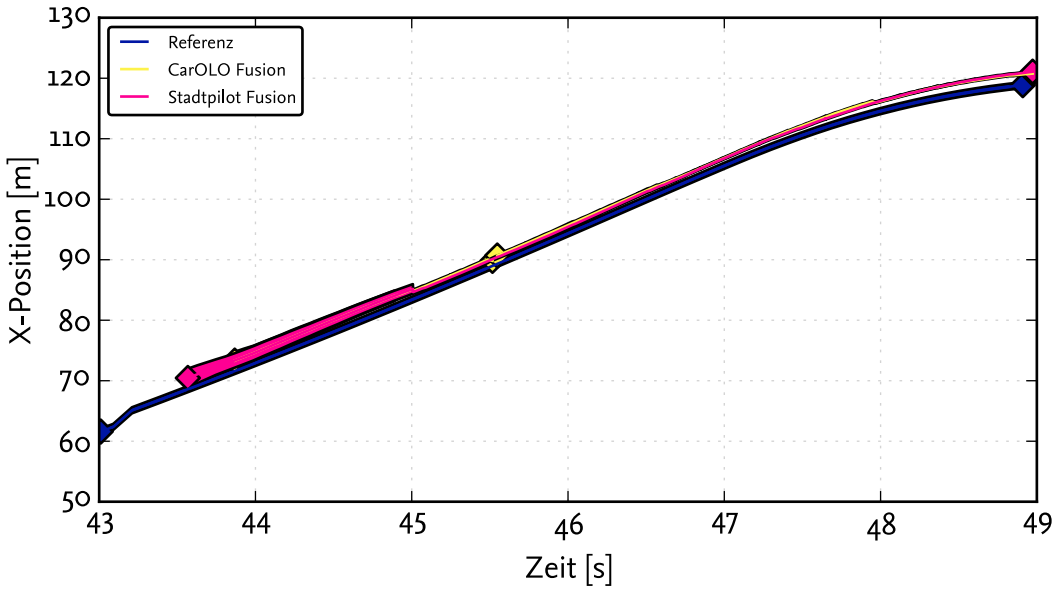


Abbildung 16.26: X-Position des Referenzfahrzeugs in Situation 2 nach Effertz (2009, Seite 137)

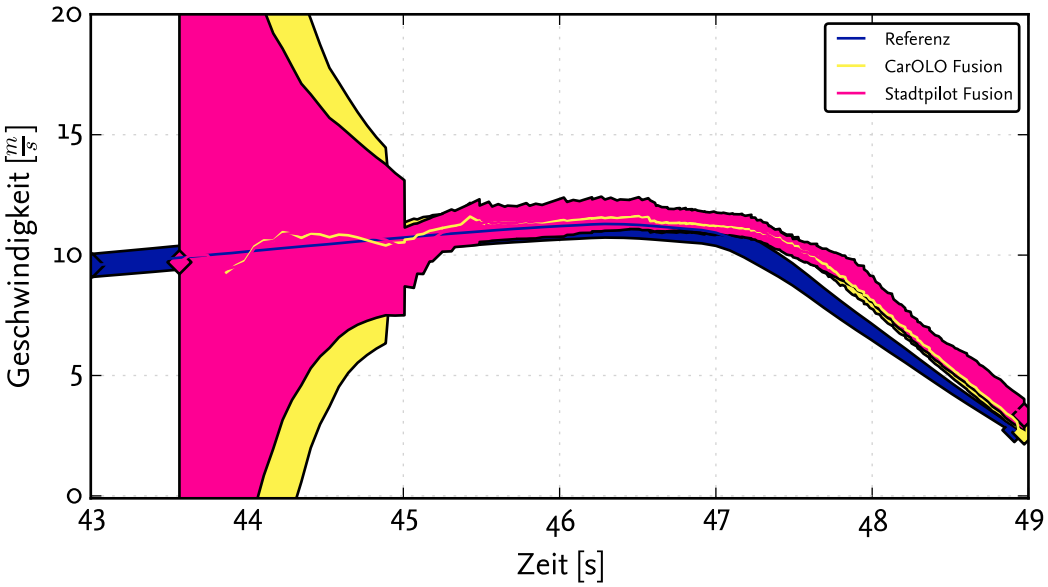


Abbildung 16.27: Geschwindigkeit des Referenzfahrzeugs in Situation 2 nach Effertz (2009, Seite 137)

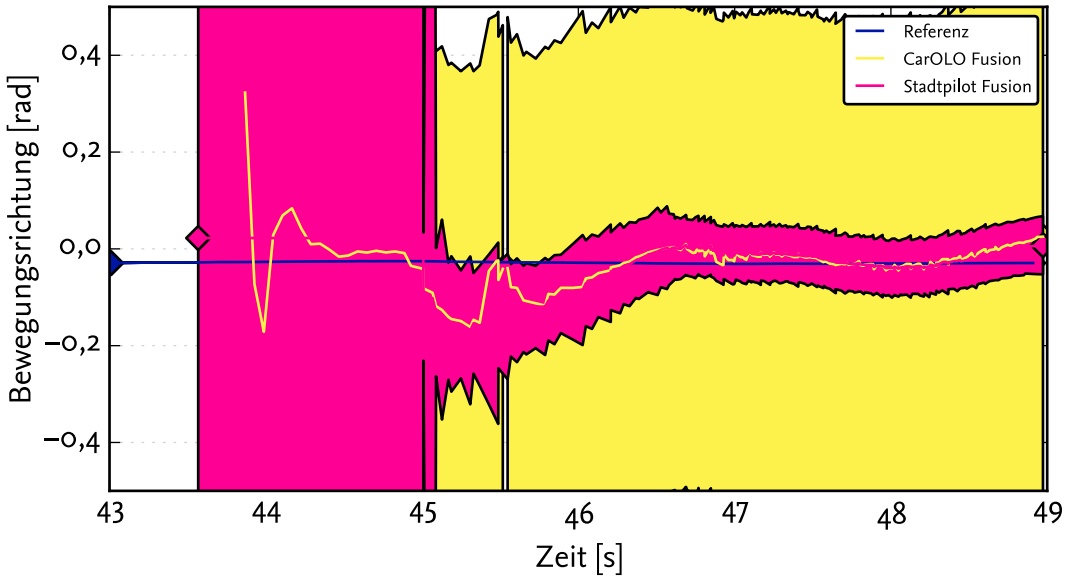


Abbildung 16.28: Bewegungsrichtung des Referenzfahrzeugs in Situation 2 nach Effertz (2009, Seite 137)

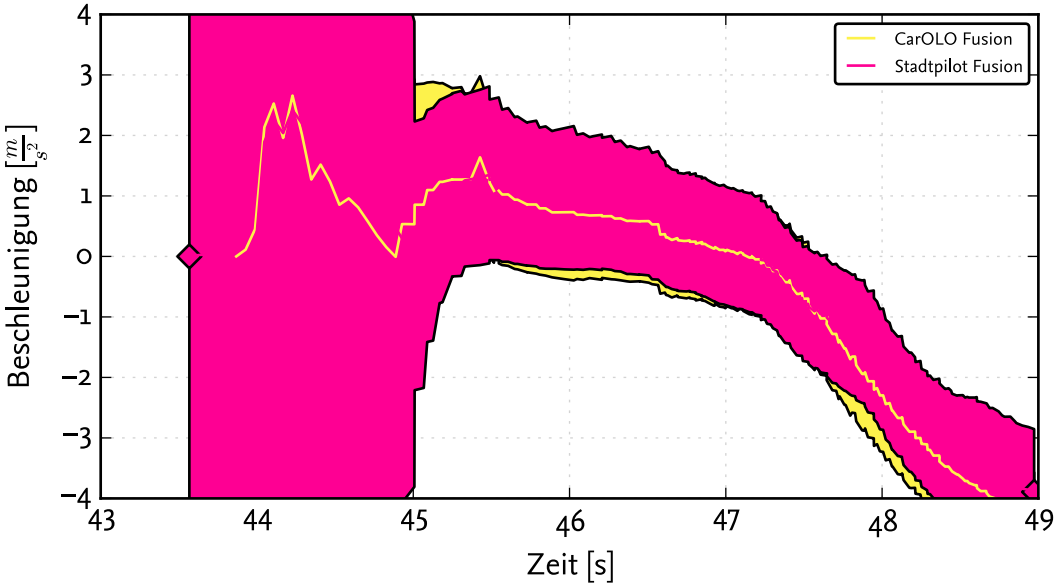


Abbildung 16.29: Beschleunigung des Referenzfahrzeugs in Situation 2 nach Effertz (2009, Seite 137)

Fahrt in Schlangenlinien

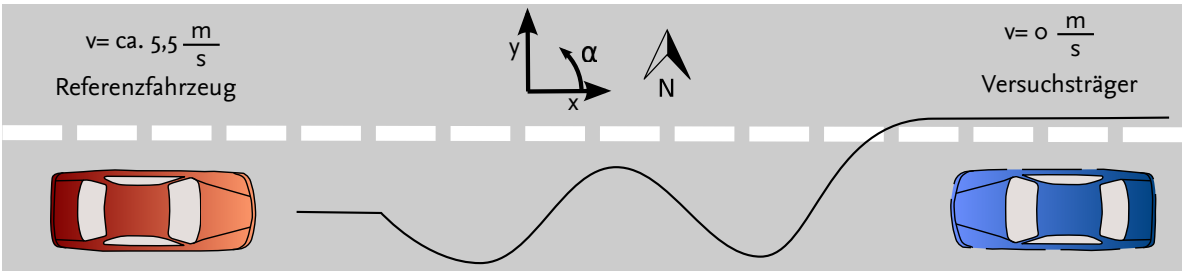


Abbildung 16.30: Situation 3 nach Effertz (2009, Seite 139f): Fahrt in Schlangenlinien

In der dritten Versuchsanordnung bewegt sich das Referenzfahrzeug in Schlangenlinien auf den Versuchsträger zu (siehe Abbildung 16.30). Dabei bewegt es sich entlang der X-Achse mit ca. 5,5 $\frac{m}{s}$und schwingt um die Y-Achse des Weltkoordinatensystems.

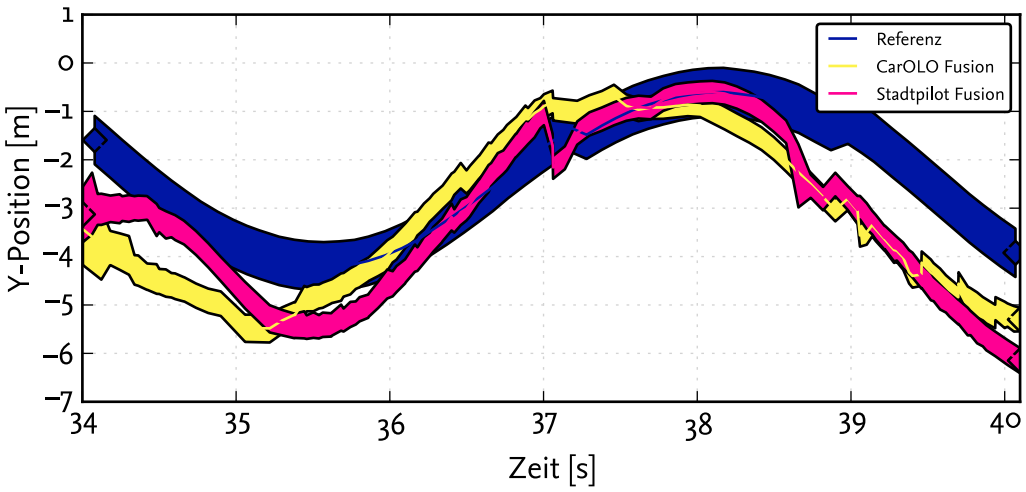


Abbildung 16.31: Y-Position des Referenzfahrzeugs in Situation 3 nach Effertz (2009, Seite 139)

Anhand von Abbildung 16.31 lässt sich die Schlangenlinienfahrt nachvollziehen. Dargestellt sind ist hier die Y-Position der Tracks der beiden Fusionen. Auffällig ist, dass die Fusion des Stadtpilot-Versuchsträgers zunächst eine konstante Y-Position aufweist, während die Position der Fusion des Projekts CarOLO sich parallel zum Referenzwert bewegt. Dies resultiert aus einer falsch angenommenen Geschwindigkeitsinformation, die wiederum aus einem falsch angenommen Geschwindigkeitwert des IBEO Alaska XT resultiert (siehe Abbildung 16.32). Nach dieser anfänglichen Unsicherheit folgen die Objekthypothesen jedoch den Referenzwerten. Betrachtet man das Geschwindigkeitssignal der Objekthypothese des Projekts CarOLO, so ist ein schwingendes Verhalten sichtbar, welches nicht in diesem Ausmaß in den Referenzdaten sichtbar ist und auch nicht durch die Stadtpilot-Fusion nachempfunden wird.

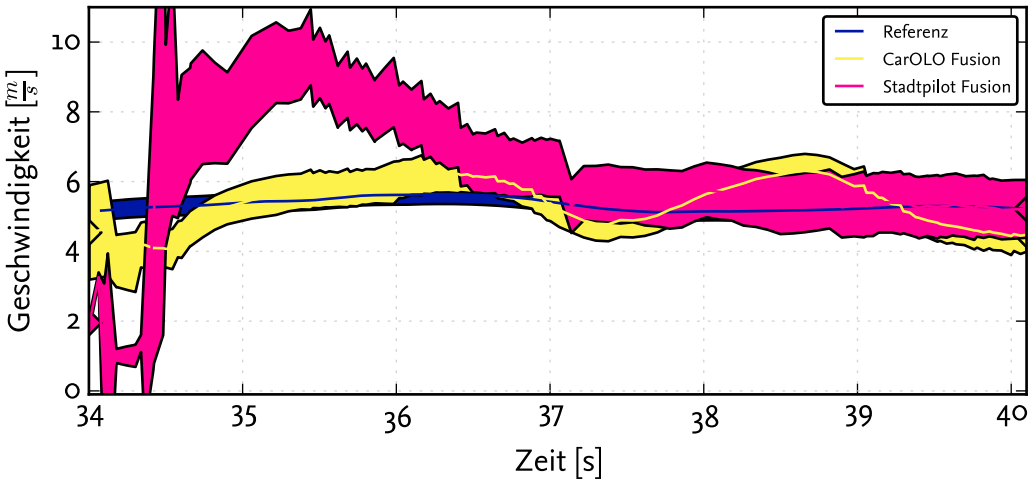


Abbildung 16.32: Geschwindigkeit des Referenzfahrzeugs in Situation 3 nach Effertz (2009, Seite 139)

16.3.1 Grenzen der objekthypothesenbasierten Fusionsmodule

In den vorangegangenen Abschnitten wurden unterschiedliche Situationen auf einem Testgelände oder im Einsatzgebiet des Projekts Stadtpilot beschrieben. Es konnte gezeigt werden, dass sich das System bis auf kurze Unterbrechungen den Anforderungen entsprechend verhält. Dennoch gibt es Situationen, die die Fusionsmodule nicht optimal handhaben können. So zeigt beispielsweise Abbildung 16.5 eine Verzögerung des Geschwindigkeitssignals bedingt durch eine fehlerhafte Modellvorstellung des vorderen Radarsensors während dieser Extremsituation. Da das Messdatum dieses Sensors aber für Normalsituationen eine verlässliche, qualitativ hochwertige Information darstellt, kann sie in diesem Fall nicht ignoriert werden und verschlechtert so das Fusionsergebnis. Mit dem derzeitigen Aufbau des Fusionsmoduls lässt sich dieses Problem nicht vermeiden. Eine Möglichkeit, diesem entgegen zu treten, wäre situationsbezogen die Filterparameter zu verändern (z. B. durch einen IMM-Filter). Ob dies in den gewünschten Zeiträumen möglich ist oder die Latenz nur weiter vergrößern würde, kann hier nicht beantwortet werden.

Das eingesetzte Objekthypothesenmodell bildet die üblicherweise auf dem Braunschweiger Stadtring auftretenden Situationen gut ab. In jedoch eher unüblichen Bewegungsmustern, wie z. B. in der Schlangenlinienfahrt aus Abbildung 16.30, stößt das Modell an seine Grenzen. Die sinusförmige Bewegung lässt sich mit dem CT-Modell nicht optimal abbilden. Darüber hinaus sorgen die häufigen Konturwechsel für zusätzliche Abweichungen.

Die Bewegung einer Objekthypothese wird in ihrem Schwerpunkt geschätzt. Diese Annahme führt im Falle von Konturwechseln zu kurzzeitigen Abweichungen des Geschwindigkeitssignals einer Objekthypothese (siehe Abbildung 16.10). Für den klassifizierten Fall ließen sich diese Abweichungen durch eine explizite Transformation des Schwerpunkts reduzieren. Eine vollständige Vermeidung ließe sich wohl nur durch ein individuelles Tracking jedes Konturstützpunkts erreichen, was aufgrund der Echtzeitanforderungen unrealistisch erscheint.

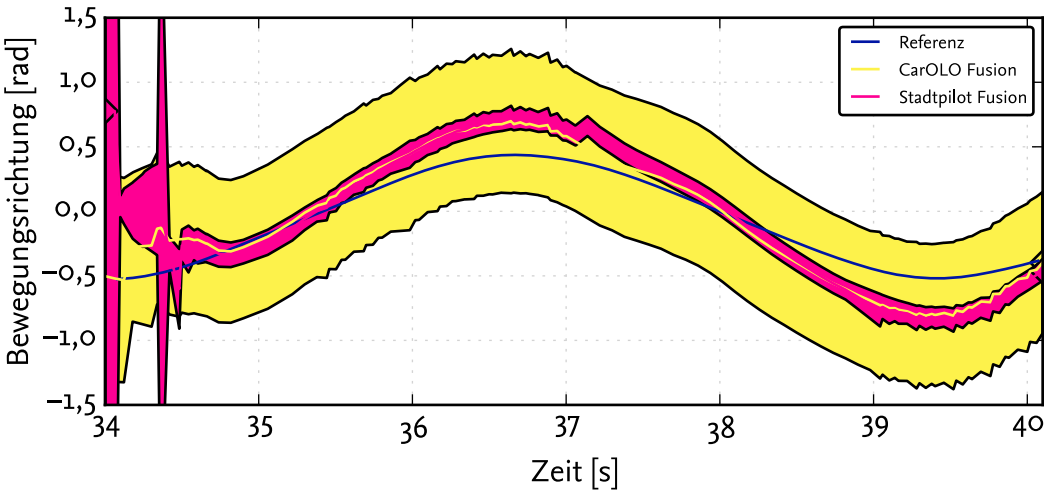


Abbildung 16.33: Bewegungsrichtung des Referenzfahrzeugs in Situation 3 nach Effertz (2009, Seite 139)

16.4 Gitterbasierte Fusion

Die Bewertung des gitterbasierten Fusionsmoduls teilt sich in mehrere Bereiche auf. Zunächst wird auf die Ergebnisse der gitterbasierten Datenfusion an sich eingegangen. Darauf folgt die Vorstellung der Ergebnisse der einzelnen *View*-Komponenten. Ein direkter Vergleich mit dem Projekt CarOLO, wie bei der objekthypothesenbasierten Fusion, ist bei der gitterbasierten Datenfusion nicht möglich, da im Projekt Stadtpilot eine andere Beschreibungsform gewählt wurde sowie andere Sensoren eingesetzt wurden.

Die Gitterdatenstruktur des Fusionsmoduls beschreibt die Wahrscheinlichkeit, dass eine Zelle durch ein Objekt belegt ist. Die Gitterstruktur besteht dabei aus 8×8 *GridBlocks à* 64×64 Zellen. Eine Zelle beschreibt ein Quadrat mit einer Kantenlänge von $0,2m$. Insgesamt deckt die Gitterstruktur eine Fläche von $10485,76m^2$ ab und wird in Gitterkoordinaten innerhalb des Weltkoordinatensystems beschrieben.

In den Abbildungen 16.34 und 16.35 ist eine statische Szene auf dem Stadtpilot-Testgelände dargestellt. Dabei befindet sich das Referenzfahrzeug ca. $10m$ vor dem Versuchsträger. In Darstellung 16.34 wurde die Referenzposition über der Gitterstruktur dargestellt. Zu sehen ist, dass die Gitterdaten und das Referenzfahrzeug weitestgehend deckungsgleich sind. Neben dem Referenzfahrzeug wird auch die Umgebungsbebauung durch die Gitterstruktur abgebildet. Vergleicht man die herausgebildeten Kanten mit den Außenkanten der Gebäude auf dem unterlegten Luftbild, so entsprechen diese den Außenkanten der Gebäude und werden somit als belegt wahrgenommen (Anforderung SPFA26).

Das gitterbasierte Fusionsmodul zeigt auch im realen Verkehr gute Ergebnisse. Die Abbildungen 16.36 und 16.37 zeigen eine Fahrt auf dem Braunschweiger Stadtring. Deutlich fällt hierbei die Freiheit der Fahrbahn auf. Durch die Abstraktionsvorschrift der Sensordaten im *SensorModel* wird der umgebende Verkehr, während er in Bewegung ist, nicht erfasst. Die Gebäude und stehende Objekte sind jedoch deutlich zu erkennen (Anforderung SPFA31).

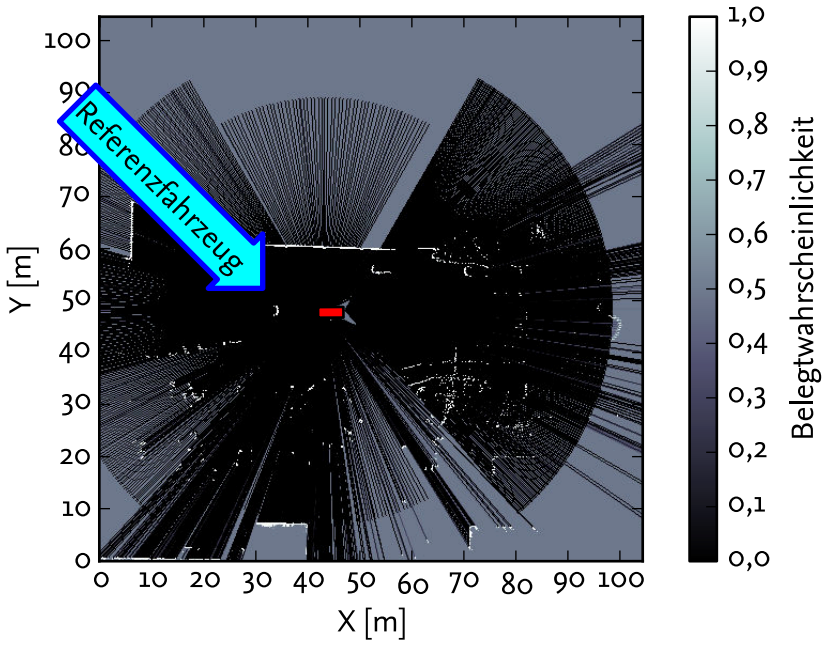


Abbildung 16.34: Gitterdatenstruktur in einer statischen Szene mit Referenzfahrzeug und Luftbild. Versuchsträger (rotes Rechteck) (Kartengrundlage: ©Stadt Braunschweig, Abteilung Geoinformationen (Nr. 011/2010)

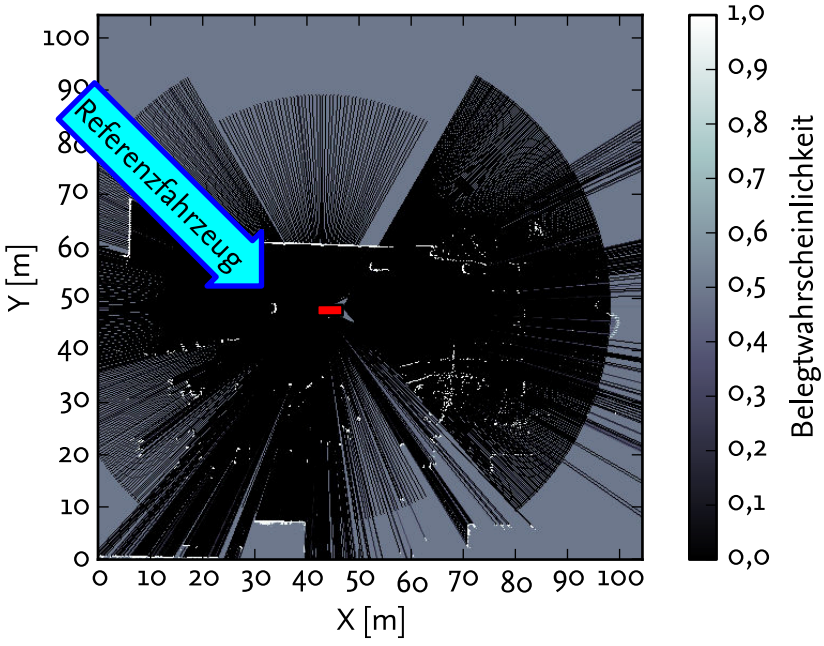


Abbildung 16.35: Gitterdatenstruktur in einer statischen Szene mit Referenzfahrzeug ohne Luftbild. Versuchsträger (rotes Rechteck)

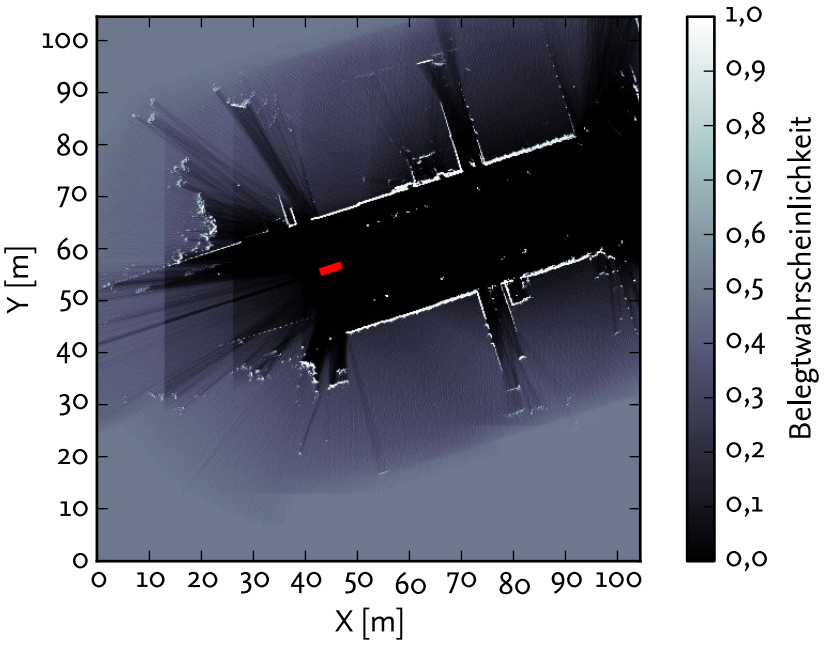


Abbildung 16.36: Gitterdatenstruktur während der Fahrt auf dem Braunschweiger Stadtring mit Luftbild. Versuchsträger (rotes Rechteck) (Kartengrundlage: ©Stadt Braunschweig, Abteilung Geoinformationen (Nr. 011/2010)

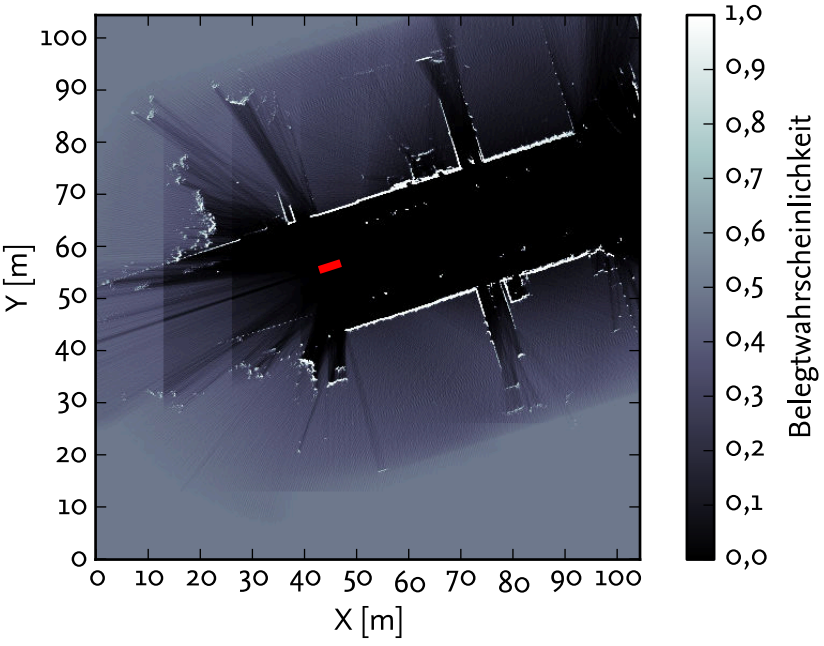


Abbildung 16.37: Gitterdatenstruktur während der Fahrt auf dem Braunschweiger Stadtring ohne Luftbild. Versuchsträger (rotes Rechteck)

16.4.1 Schätzung der Fahrbahnbreite

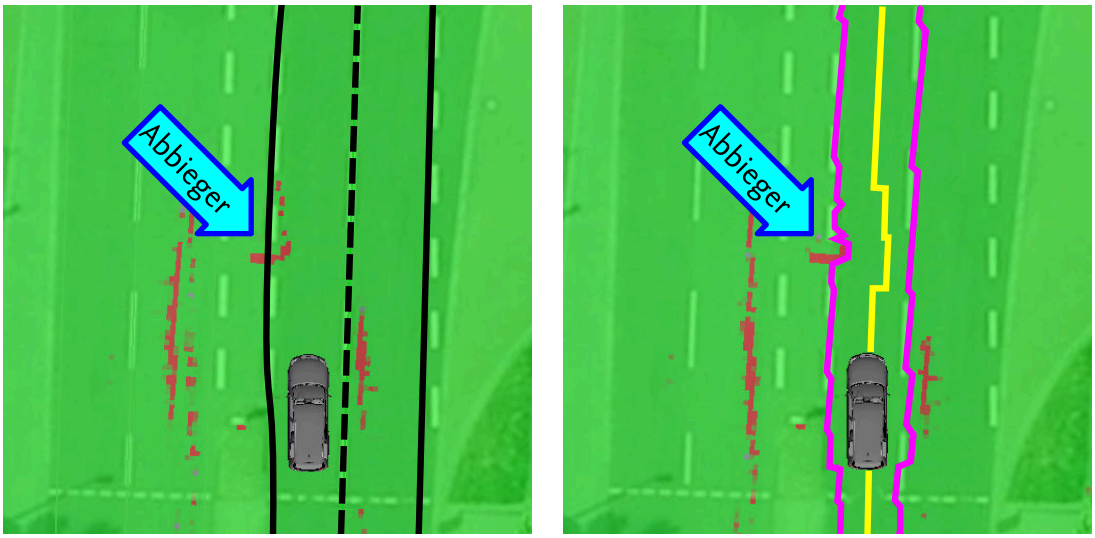


Abbildung 16.38: Schätzung der Fahrbahnbreite anhand von Gitterdaten. Ergebnis des Schätzalgorithmus (rechts), Fahrbahnverlauf der digitalen Karte (links), belegte Zellen (rot), nicht belegte Zellen (grün) (Kartengrundlage: ©Stadt Braunschweig, Abteilung Geoinformationen (Nr. 011/2010)

Zur Schätzung der aktuellen Fahrbahnbreite wird der in Abschnitt 15.4 beschriebene Algorithmus eingesetzt. Dabei wird die aus der digitalen Karte des Fahrzeugs bekannte Fahrbahnbreite anhand der Daten der gitterbasierten Sensordatenfusion eingeschränkt (Anforderung SPFA26). Der Algorithmus bestimmt bis zu einem Abstand von $40m$ vor dem Versuchsträger die aktuelle Fahrbahnbreite. In der in Abbildung 16.38 dargestellten Situation schränkt ein anderer Verkehrsteilnehmer den Fahrstreifen des Versuchsträgers während eines Linksabbiege-Manövers ein. In der linken Darstellung sind die Gitterdaten zusammen mit dem Fahrbahnverlauf der digitalen Karte abgebildet. Das Heck des stehenden wartenden Abbiegers ist in der Gitterdatenstruktur deutlich zu erkennen und befindet sich erheblich innerhalb des eigenen Fahrstreifens. Auf der rechten Seite ist das Ergebnis des Algorithmus zu sehen. In Violett ist hier eine diskretisierte Form der Fahrstreifengrenzen zu sehen. Die Diskretisierung der Grenzen entspricht der Auflösung der Gitterdatenstruktur. Während der Versuchsträger auf Basis der digitalen Karte auf der linken Seite dem Fahrstreifenverlauf folgen würde, so ist auf der rechten Seite die Verengung durch den Abbieger zu erkennen. Der verdeckte Bereich des Fahrstreifens ist durch Abweichung der gelben Linie von der Fahrstreifenmitte in der rechten Abbildung mit einer Diskretisierung von $2m$ dargestellt.

16.4.2 Bestimmung des Erfassungsbereichs der Sensoren

Das Wissen darüber, ob ein Hindernis wahrgenommen werden kann, ist gerade bei Abbiegemanövern von großer Bedeutung. Verdeckt beispielsweise ein Hindernis einen großen Teil der Kreuzung, so scheint ein Abbiegemanöver möglich, obwohl weitere Fahrzeuge durch Hindernisse verdeckt seien könnten. Um dieses Problem zu beheben, wurde in Abschnitt 15.5 ein Algorithmus vorgestellt, der auf Basis einer Gitterdatenstruktur einen Wert für die

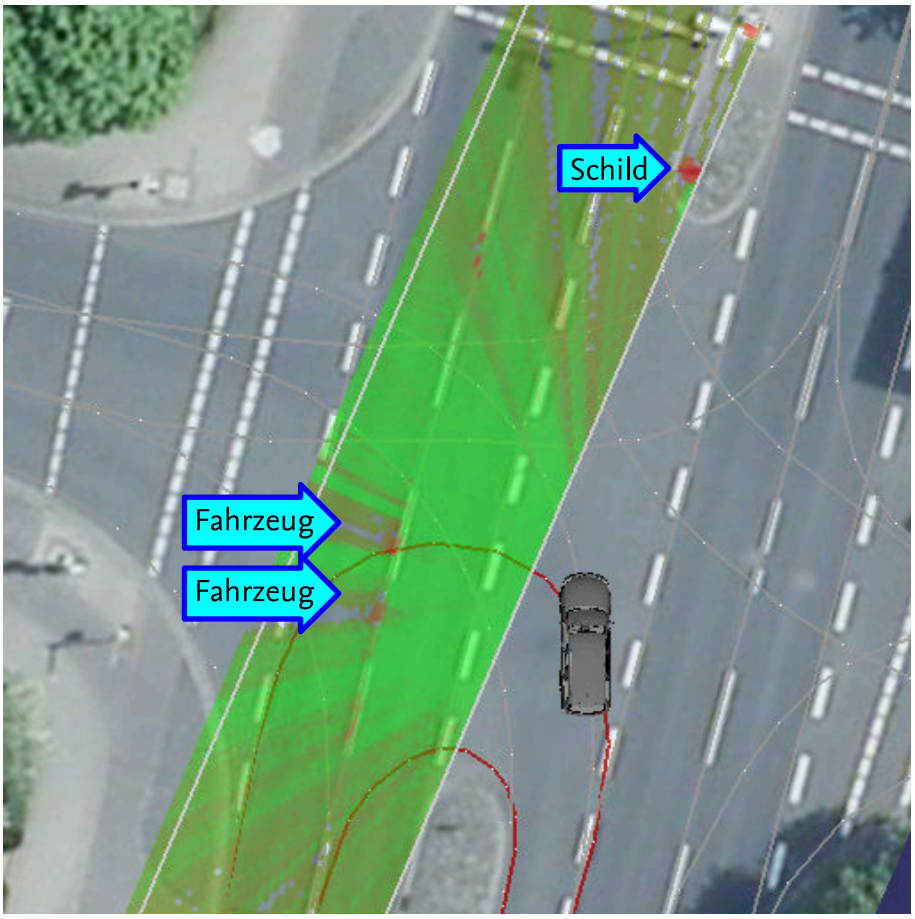


Abbildung 16.39: Sensorabdeckung während eines Abbiegemanövers. Gitterdatenstruktur (Grün/Rot-Fläche), belegte Zellen (rot), nicht belegte Zellen (grün), Fahrstreifengrenzen (rote und graue Linien) (Kartengrundlage: ©Stadt Braunschweig, Abteilung Geoinformationen (Nr. 011/2010))

aktuelle Abdeckung der Umgebung durch die Sensoren berechnet. In Abbildung 16.39 ist eine Szene während eines Abbiegemanövers dargestellt. Auf Basis der digitalen Karte des Versuchsträgers wurde die Gitterdatenstruktur über die *LayerConfiguration*-Schnittstelle in Größe und Lage dem für die Applikation relevanten Bereich angepasst. Da die eingesetzte ortsfeste Gitterdatenstruktur ein nach Norden ausgerichtetes Rechteck repräsentiert, treten hierbei Diskretisierungsfehler auf. In diesem Fall ist eine leichte Drehung sowie eine leichte Vergrößerung der Struktur gegenüber dem durch die Applikation gewünschten Bereich zu sehen. Nach der Anpassung der Datenstruktur bilden sich, bedingt durch das Lasersensormodell, belegte und nicht belegte Bereiche heraus. Die zur Bestimmung des aktuellen Erfassungsbereichs relevanten Bereiche bilden sich durch die unentschiedenen Bereiche (transparent). Auf der Abbildung zeigt sich diese Eigenschaft vor allem bei einem Fahrzeug (links) und einem Schild (oben). Hierbei lässt sich deutlich der transparente Bereich hinter den Hindernissen erkennen. Dieser kann aktuell nicht durch einen Sensor wahrgenommen werden und verhindert so möglicherweise den Beginn eines Abbiegemanövers (Anforderung SPFA29).

16.4.3 Konstruktion von Objekthypothesen aus Gitterdaten

In der Gitterdatenstruktur bilden sich, wie auf Seite 162 erwähnt, vor allem statische Objekte heraus (Anforderung SPFA31). Diese Eigenschaft kann dazu genutzt werden, statische Objekthypothesen zu bilden und so eine weitere Datenquelle für die objekthypothesenbasierte Fusion bereitzustellen. In Abschnitt 15.6 wurde hierzu ein Algorithmus vorgestellt, welcher auf Basis eines 2D-Split&Merge-Verfahrens Objekthypothesen aus Gitterdaten erzeugt. Da dieser Algorithmus nur in Matlab vorliegt, wurden die Gitterdaten mithilfe einer *View*-Komponente in ein Bitmap überführt, von der Matlab-Umsetzung verarbeitet und anschließend wieder als *UnifiedSensorObjectList* zur weiteren Verarbeitung eingelesen.

Zur Demonstration des 2D-Split&Merge-Verfahrens wird auf die Szenen aus den Abbildungen 16.34 und 16.36 zurückgegriffen. In der ersten Szene wird das Referenzfahrzeug unmittelbar vor dem Versuchsträger in einer statischen Szene betrachtet. In Abbildung 16.40 sind die gebildeten Objekthypothesen mit Kontur-Geometriemodell als rote Konturzüge über einem Luftbild dargestellt. Abbildung 16.41 zeigt dagegen nur die Polygonzüge ohne unterlegtes Luftbild. Zu sehen ist, dass das Verfahren die Kante des oberen Gebäudes deutlich abbildet. Der untere Bewuchs kann nicht einheitlich abgebildet werden, da es hier keine klare äußere Kontur gibt.

Zur Bewertung der Kontur des Referenzfahrzeugs wird das Ergebnis des Algorithmus mit der Objekthypothesenkontur des IBEO Alaska XT-Sensors verglichen (siehe Abbildung 16.44). Als Eingangsdaten des gitterbasierten Fusionsmoduls wird an dieser Stelle ausschließlich auf die Messpunktwolke des IBEO Alaska XT zurückgegriffen. Somit nutzen beide Objekthypothesenbildungsalgorithmen die gleiche Datengrundlage. Beide Algorithmen zur Objekthypothesenerzeugung bilden die hintere Kontur des stehenden Referenzfahrzeugs ab. Aus diesem Grund sind beide im Vergleich zu der Darstellung des Referenzdatums leicht gekrümmt. Die durch den IBEO Alaska XT gebildete Objekthypothese ist leicht gegenüber der Referenz nach hinten verschoben, während der in Abschnitt 15.6 vorgestellte Algorithmus diese Verschiebung nicht aufweist. Vergleicht man die beiden Konturen mit der in Abschnitt 16.1.2 vorgestellten Berechnung des MSE, so zeigt sich auch hier ein leicht schlechterer Wert für die Objekthypothesenerzeugung des IBEO Alaska XTs:

Objekthypothesenerzeugung	MSE
IBEO Alaska XT	0.2399 m^2
2D-Split&Merge-Algorithmus	0.1339 m^2

Auch in der Szene auf dem Braunschweiger Stadtring bildet der 2D-Split&Merge-Algorithmus die Außenkanten der statischen Objekte gut ab. In den Abbildungen 16.42 und 16.43 ist die bereits bekannte Fahrt auf dem Braunschweiger Stadtring zu sehen. Während Abbildung 16.42 die Polygonzüge der Objekthypothesen über dem zugehörigen Luftbild zeigt, stellt Abbildung 16.43 die Objekthypothesen ohne Hintergrund dar.

Übersicht über die erfüllten Anforderungen

Die vorangegangenen Abschnitte lassen sich in folgender Tabelle zusammenfassen:

Anforderung	Erfüllt		Anforderung	Erfüllt
SPFA26	✓		SPFA29	✓
SPFA31	✓			

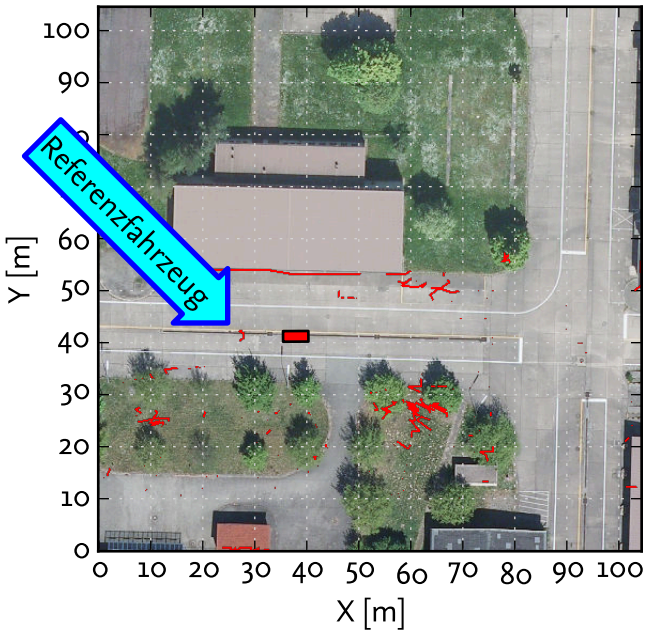


Abbildung 16.40: Erzeugung von Objekthypothesen aus einer Gitterdatenstruktur in einer statischen Szene mit Referenzfahrzeug und Luftbild. Versuchsträger (rotes Rechteck) (Kartengrundlage: ©Stadt Braunschweig, Abteilung Geoinformationen Nr. 011/2010)

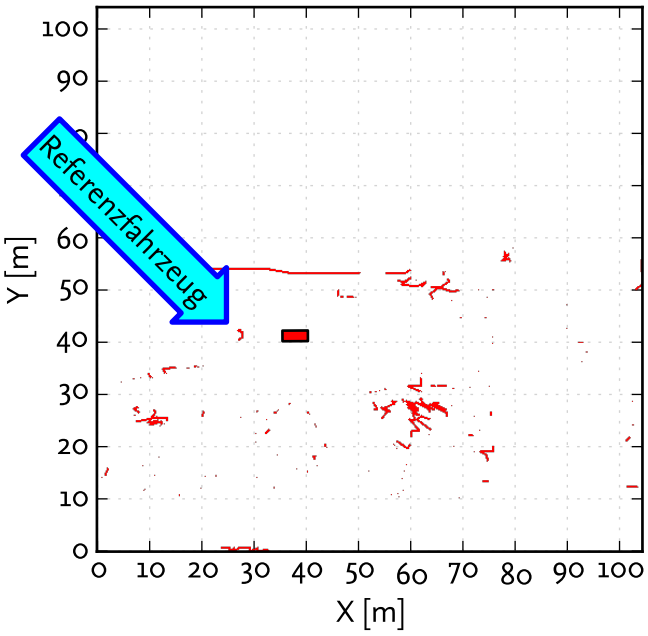


Abbildung 16.41: Erzeugung von Objekthypothesen aus einer Gitterdatenstruktur in einer statischen Szene mit Referenzfahrzeug ohne Luftbild. Versuchsträger als rotes Rechteck

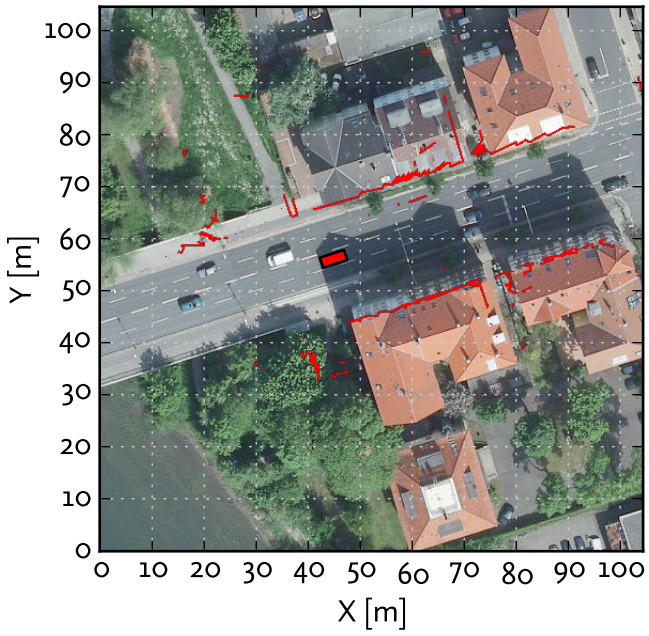


Abbildung 16.42: Erzeugung von Objekthypothesen aus einer Gitterdatenstruktur während der Fahrt auf dem Braunschweiger Stadtring mit Luftbild. Versuchsträger (rotes Rechteck) (Kartengrundlage: ©Stadt Braunschweig, Abteilung Geoinformationen Nr. 011/2010)

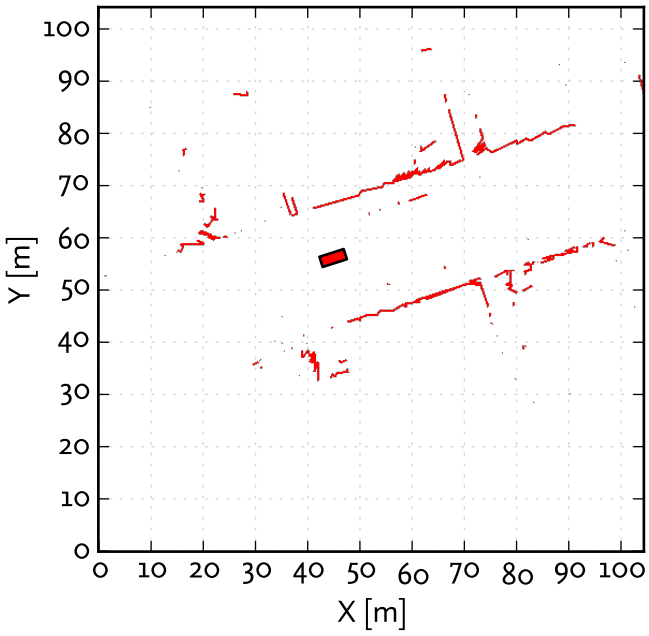


Abbildung 16.43: Erzeugung von Objekthypothesen aus einer Gitterdatenstruktur während der Fahrt auf dem Braunschweiger Stadtring ohne Luftbild. Versuchsträger (rotes Rechteck)

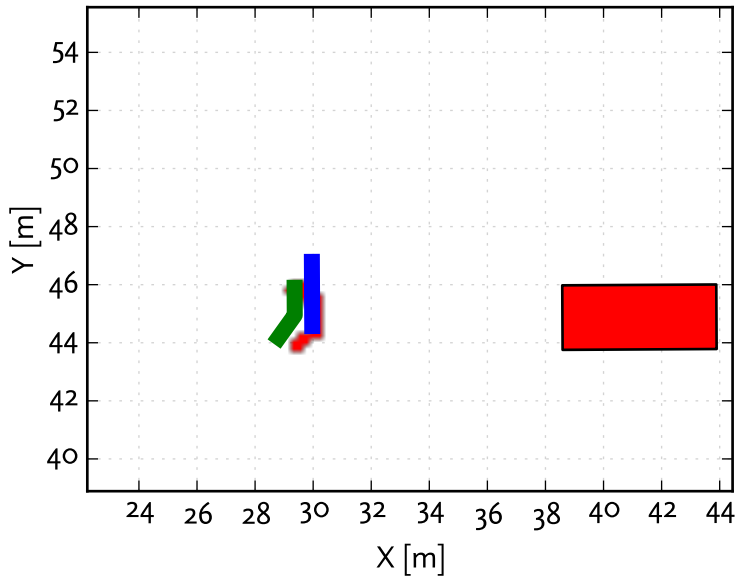


Abbildung 16.44: Detailansicht der Abbildung des Referenzfahrzeugs (blau), des Ergebnisses des Algorithmus (rot) und der Objektbildung des IBEO Alaska XTs (grün), Versuchsträger (rotes Rechteck)

16.4.4 Grenzen der Umsetzung des gitterbasierten Fusionsmoduls

Das vorgestellte gitterbasierte Fusionsmodul stellt eine übliche Realisierung für eine Sensordatenfusion mit Lasersensor und binärem Bayes-Filter dar. Da die Demonstration der Funktionsfähigkeit der Softwarearchitektur im Vordergrund stand, setzt es keine wesentlichen neuen Erkenntnisse im Forschungsbereich der gitterbasierten Datenfusion um. Für die umgesetzten *View*-Komponenten ist die erreichte Datengüte ausreichend.

Durch entsprechende Parametrierung des Sensormodells und der Gitterdatenstruktur werden hier im Wesentlichen statische Hindernisse in der Gitterdatenstruktur abgebildet. In Situationen, in denen der Verkehr jedoch insgesamt zum Stehen kommt, finden sich auch die Konturen nicht dauerhaft statischer Objekte in der Gitterdatenstruktur. Dies könnte beispielsweise durch eine Komponente zur Dynamikklassifikation (Matthaei u. a., 2011) oder durch eine Rückführung der Objekthypothesen des *MainFusionModules* gelöst werden.

Die Schätzung der Fahrbahnbreite liefert gute Ergebnisse, solange sich der Versuchsträger bewegt. Kommt er und der umliegende Verkehr jedoch beispielsweise an einer Lichtsignalanlage zum Stehen, so schränken die Fahrzeuge vor dem Versuchsträger den unbelegten Bereich ein. Ist dabei auch die Mitte des eigenen Fahrstreifens belegt, so ist der Algorithmus nicht mehr anwendbar, da bereits die inertiale Zelle belegt ist. In diesem Fall muss der Versuchsträger warten, bis der Bereich durch eine ausreichende Zahl an Messungen erneut als unbelegt wahrgenommen wird.

Die Bestimmung des Erfassungsbereichs stellt einen ersten Schritt für eine Entscheidung über ein Linksabbiege-Manöver dar. Bis zur endgültigen Realisierung eines solchen Manövers auf dem Braunschweiger Stadtring ist jedoch noch weiterer Forschungsbedarf vorhanden.

Zur endgültigen Nutzung des Algorithmus zur Rekonstruktion von Objekthypothesen ist eine Transformation des Programmcodes von Matlab nach C++ notwendig. Da keine

aufwändigen Operationen durchgeführt werden, sollte diese Umsetzung echtzeitfähig sein. Bedingt durch die Diskretisierung der Gitterdatenstruktur löst der Algorithmus die Stützpunkte der Objekthypothesen nur mit der Auflösung der Gitterdaten auf und erzeugt damit einen Fehler in der Größenordnung der Gitterauflösung. Darüber hinaus hat sich der χ^2-Test mit 5% in den gezeigten Anwendungsfällen zwar als sinnvoll erwiesen, je nach Qualität der Sensoren und dem Fahrzeugumfeld muss dieser Wert angepasst werden. Eine Bewertung der Größe des Rauschens der Gitterdatenstruktur könnte hier einen Anhaltspunkt bieten.

17 Zusammenfassung

Der Teil über die Realisierung der Softwarearchitektur stellte ausgewählte Algorithmen aus der Umfeldwahrnehmung des Projekts Stadtpilot vor. Zunächst wurden projektspezifische Anforderungen definiert. Diese bilden die Grundlage für die Entwurfsentscheidungen und die Auswahl der Algorithmen. Anschließend wurden insgesamt drei Fusionsmodule als Umsetzung der Elemente des *Sensor Data Fusion Layers* beschrieben: zwei objekthypothesenbasierte Fusionsmodule und ein gitterbasiertes Fusionsmodul.

Beginnend mit der objekthypothesenbasierten Fusion wurde das konturklassifizierende Kalman-Filter vorgestellt. Hierbei wurde zunächst das Geometrie- und Bewegungsmodell des Filters festgelegt und anschließend auf die Konturpunktverarbeitung eingegangen. Da sich die Beschränkung auf nur ein Geometrie- sowie ein Bewegungsmodell nicht als ausreichend erwiesen hat, werden diese anhand eines Dempster-Shafer-Filters zur Laufzeit bestimmt. Für das Bewegungsmodell bedeutet das, dass in jedem Filterschritt bestimmt wird, ob das CV- oder das CT-Modell angewendet wird. Diese Unterscheidung ermöglicht das Tracking sowohl von bewegten als auch von stehenden Objekthypothesen. Das Geometriemodell hingegen wird nicht auf klassische Weise mit einem Mode-Switch umgeschaltet, sondern gewinnt oder verliert, je nach Eigenschaften der eingehenden Messdaten, an Flexibilität. Dies ermöglicht eine Reduktion des Rauschanteils in der Kontur und damit eine Verbesserung der Zustandsgrößen. Zur Anpassung der Kontur nach einem Modellwechsel wurden zwei Verfahren vorgestellt. Das erste erwies sich als nicht hinreichend. Das zweite Verfahren basierte auf der Evaluation von unterschiedlichen Eigenschaften des Messwerts, die auf die Meta-Kontur abgebildet wurden.

Zur Objekthypotheseninitialisierung und zur Bestimmung der Stabilität einer Objekthypothese wurde anschließend eine darauf ausgerichtete Form des Trackings vorgestellt. Diese schätzt die Geschwindigkeit der Hypothese und bestimmt die Stabilität der Objekthypothese. Zur Entscheidung über die Initialisierung eines neuen Tracks im *MainFusionModule* wird die Redundanz der Sensorabdeckungsbereiche genutzt. Hierzu wurde ein Modul vorgestellt, das auf Basis von booleschen Ausdrücken Entscheidungen über die Zulassung von neuen Tracks in der Hauptfusion trifft.

Im Abschnitt über die gitterbasierten Fusionsmodule wurde zunächst auf die Verarbeitung von Lasermessdaten und anschließend auf das verwendete binäre Bayes-Filter eingegangen. Die Auswertung der Gitterdatenstruktur geschieht mit drei unterschiedlichen *View*-Komponenten. Hierzu wurde zunächst eine angepasste Variante des Algorithmus nach Weiss (2011, Seite 128ff) vorgestellt. Dieses Verfahren wird zur Schätzung der Fahrbahnbreite genutzt und ermöglicht es dem Versuchsträger, seine Trajektorie bei kleinen Verengungen des Fahrstreifens leicht zu verändern. Die zweite *View*-Komponente beschäftigt sich mit der Bestimmung des durch die Sensoren abdeckbaren Bereichs um das Fahrzeug. Der Algorithmus bestimmt dabei zunächst die Fläche eines möglichen Konflikts zwischen den anderen Verkehrsteilnehmern und dem Versuchsträger. Davon ausgehend wird die Gitterdatenstruktur evaluiert und in einen Wert zur Abdeckungsfläche überführt. Die letzte vorgestellte Komponente schätzt Objektkonturen aus der Gitterdatenstruktur. Dazu wird ein

2D-Split&Merge-Algorithmus eingesetzt, welcher sowohl mit Gitterdatenstrukturen umgehen kann als auch durch andere Objekte verdeckte Elemente abbildet.

An die algorithmischen Abschnitte dieses Teils schließt sich ein Abschnitt über die Leistungsparameter des Fusionssystems an. Dabei wurden die projektspezifischen Anforderungen der objekthypothesenbasierten Fusion anhand von verschiedenen Szenarien diskutiert und gezeigt, dass die Anforderungen, bis auf kurzzeitige Unterbrechungen, erfüllt werden.

Darauffolgend wurde das konturschätzende Kalman-Filter mit einem Interacting-Multiple-Model-Filter mit unterschiedlichen Geometriemodellen verglichen. Das IMM-Filter schnitt dabei schlechter als das konturschätzende Kalman-Filter ab. Dies resultierte vor allem aus der fehlenden eindeutigen Entscheidung des IMM-Filters für ein Geometriemodell.

Ein Vergleich mit dem Umfeldwahrnehmungssystem des Vorgängerprojekts CarOLO schließt den Abschnitt über die objekthypothesenbasierte Fusion ab. Dabei wurden die drei in Effertz (2009) beschriebenen Situationen mit beiden Umfeldwahrnehmungen nachgestellt und anschließend die Ergebnisse gegenübergestellt.

Nach der objekthypothesenbasierten Fusion wurden die Ergebnisse der gitterbasierten Sensordatenfusion evaluiert. Hierbei lag der Schwerpunkt auf der Darstellung der Ergebnisse durch die *View*-Komponenten. Zunächst wurde die Fahrbahnbreitenschätzung betrachtet. In der dargestellten Situation schränkte der Algorithmus die Fahrbahn entsprechend den Messdaten ein. Anschließend wurde die Schätzung des Erfassungsbereichs aus Gitterdatenstrukturen betrachtet. Abgeschlossen wurde der Abschnitt mit einer Evaluation des Algorithmus zur Konstruktion von Objekthypothesen aus Gitterdaten.

Bedingt durch die Ausrichtung dieser Arbeit auf die Umfeldwahrnehmung des Versuchsträgers werden bisher nicht alle Ergebnisse der Algorithmen im Einsatzgebiet Braunschweiger Stadtring genutzt. Die Komponenten des *Sensor Specific Layers* übernahmen die Dekodierung der sensorspezifischen Datenströme sowohl auf dem Braunschweig Stadtring als auch dem Testgelände. Für den Einsatz auf dem Stadtring konnten die Daten der Fusionsmodule bisher nur für das ACC-Modul genutzt werden. Die Ergebnisse der anderen Algorithmen wurden bisher für Tests von weiteren Fahraufgaben auf einem abgeschlossenen Testgelände eingesetzt.

Teil IV: Zusammenfassung und Ausblick

18 Zusammenfassung

Die vorliegende Arbeit unterteilt sich in zwei Abschnitte. Zunächst wurde eine Softwarearchitektur zur Umfeldwahrnehmung im Automobil vorgestellt. Anschließend wurde diese für ein konkretes Projekt umgesetzt.

Zur Definition der Softwarearchitektur wurde zunächst eine domänenspezifische Architektur bestehend aus vier Basisfunktionen vorgestellt. Diese teilt jeden Fusionsprozess in die Funktionen *Selektion*, *Abstraktion*, *Fusion* und *Datenspeicherung*. Stellt man sie geeignet zusammen, so lassen sich damit alle zur Umfeldwahrnehmung im Automobil üblichen Fusionsprozesse abbilden. Darauf sowie auf dem Architekturkontext aufbauend, wurden Anforderungen an die zu entwickelnde Softwarearchitektur beschrieben. Diese resultieren einerseits aus dem allgemeinen Fusionsprozess und andererseits aus den Erfahrungen des Vorgängerprojekts.

Die konkrete Softwarearchitektur teilt sich in drei Softwarepakete auf:

- *Sensor Specific Layer*
- *Sensor Data Fusion Layer*
- *Application Layer*

Der *Sensor Specific Layer* bindet Umfeld wahrnehmende Sensoren an die Architektur an. Er implementiert konkrete Sensorprotokolle und konvertiert die enthaltenen Daten in zwei Datenformate, auf denen die weitere Verarbeitung aufbaut. Innerhalb des *Sensor Data Fusion Layers* werden Sensordatenfusionen in Fusionsmodulen zusammengefasst. Diese bilden je eine konkrete objekthypothesen- oder gitterbasierte Sensordatenfusion ab. Der *Application Layer* bildet die Datensenke für die Fusionsmodule. In ihm werden alle konsumierenden Anwendungen (z. B. eine Fahrerassistenzanwendung oder eine automatische Fahrzeugführung) zusammengefasst.

Auf die einzelnen Pakete wurde im Weiteren detailliert eingegangen. Jedes Paket wurde aus der Baustein- sowie der Laufzeitsicht betrachtet. Dabei wurden zwei getrennte Architekturen für objekthypothesen- und gitterbasierte Fusionsmodule aufgestellt. Während der Zusammenstellung der Komponenten und Klassen wurden die aufgestellten Anforderungen als Grundlage zur Bewertung und Auswahl genutzt. Anschließend wurden die allgemeinen Anforderungen mithilfe von Wahr-Unwahr-Entscheidungen und der Auswertung von Messdaten bewertet.

Zur Anwendung kam die Architektur im Projekt Stadtpilot. Dieses hat sich zum Ziel gesetzt, mit dem Versuchsträger Leonie den Braunschweiger Stadtkern automatisch zu umrunden. Dazu ist ein Umfeldwahrnehmungssystem notwendig. Zum Einsatz kommt sowohl eine objekthypothesen- als auch eine gitterbasierte Sensordatenfusion. Um diese näher zu beschreiben, wurden zunächst Anforderungen an die Umfeldwahrnehmung aufgestellt. Im Vergleich zu den Anforderungen an die Architektur wurden hier projektspezifische Aspekte berücksichtigt.

Die objekthypothesenbasierte Fusion teilt sich wiederum in zwei getrennte Fusionsmodule auf. Das *MainFusionModule* enthält ein konturschätzendes Kalman-Filter. Dieses basiert auf den Erfahrungen des Teams CarOLO der DARPA Urban Challenge und beschreibt Ob-

jekthypothesen durch offene Polygonzüge. Dieses Verfahren hat sich jedoch bei den höheren Geschwindigkeiten der Verkehrsteilnehmer des Braunschweiger Stadtrings als zu anfällig gegenüber Konturrauschen erwiesen. Aus diesem Grund wurde es um eine Konturschätzung mithilfe der Evidenztheorie erweitert. Auf diese Weise werden auf Basis der eingehenden Messdaten einfache Konturen (Meta-Konturen) geschätzt und das Objekthypothesenmodell entsprechend angepasst. Das zweite Fusionsmodul zielt auf die Bestimmung von stabilen Objekthypothesen ab. Um dem *MainFusionModule* von Anfang an ein stabiles Tracking zu ermöglichen, werden noch unbekannte Objekthypothesen zunächst diesem Modul zugeführt. Verhält sich eine Objekthypothese über einen gewissen Zeitraum modellkonform, so wird sie dem *MainFusionModule* zum fortwährenden Tracking übergeben.

Als gitterbasiertes Fusionsmodul kommt ein Modul mit binärem Bayes-Filter zum Einsatz. Die Ergebnisse werden über unterschiedliche Sichten auf die Gitterdatenstruktur der Applikation bereitgestellt. Hierzu wurden drei unterschiedliche Algorithmen implementiert. Der Versuchsträger besitzt eine sehr detaillierte statische Karte der Versuchsstrecke. Da sich diese aber, bedingt durch andere Verkehrsteilnehmer, laufend ändert, muss sie durch aktuelle Messwerte ergänzt werden. Dazu wurde das Verfahren nach Weiss (2011, Seite 128ff) um die Möglichkeiten des Stadtpilot-Fahrzeugs erweitert. Mithilfe der digitalen Straßenkarte und der Gitterdatenstruktur kann so der befahrbare Bereich der Fahrbahn bestimmt werden. Ferner lassen sich die Daten des Fusionsmoduls zur Berechnung des aktuellen Erfassungsbereichs der Sensoren nutzen. Dieser kann beispielsweise die Grundlage für eine Entscheidung über ein Abbiegemanöver darstellen. Die letzte Sicht auf die Gitterdatenstruktur stellt ein Algorithmus zur Rekonstruktion von Objekthypothesen dar. Mithilfe eines 2D-Split&Merge-Algorithmus werden die Außenkanten von sich im Abdeckungsbereich der Gitterdatenstruktur abbildenden Objekten als offene Polygonzüge beschrieben. Diese könnten an die objekthypothesenbasierte Sensordatenfusion übermittelt werden und so die Daten der anderen Sensoren während des Trackings stützen.

Daran angeschlossen ist ein Evaluationsteil der projektspezifischen Anforderungen an die objekthypothesenbasierte Sensordatenfusion. Hierbei wurden zunächst Messfahrten analysiert und anschließend das konturschätzende Kalman-Filter mit einem IMM-Filter verglichen. Der Evaluationsteil zur gitterbasierten Sensordatenfusion widmet sich vor allem den unterschiedlichen Sichten auf die Gitterdatenstruktur. Anhand unterschiedlicher Fahrszenen konnte so die Funktionsfähigkeit gezeigt werden.

Mit der Entwicklung der Softwarearchitektur zur Fusion von Umfeld wahrnehmenden Sensoren ist der Grundstein für eine einfache Weiterentwicklung der Sensordatenfusionsalgorithmen gelegt. Ihr Einsatz könnte in den weiteren Arbeiten des Instituts fortgeführt werden, um die einfache Entwicklung neuer Algorithmen zu ermöglichen. So ließe sich das konturschätzende Kalman-Filter beispielsweise in zwei Richtungen weiterentwickeln. Zunächst könnte das Filter um eine Existenzschätzung nach Mählisch (2009) erweitert werden. Durch diese Änderung könnten einige Heuristiken im Fusionsablauf auf statistische Grundlagen gestellt werden. Eine weitere Möglichkeit stellt die Anpassung der Meta-Konturen dar. Vorstellbar wäre hier der Einsatz ganz anderer, sehr viel spezifischerer Objekthypothesenmodelle für bestimmte Verkehrsteilnehmer. Im Bereich der gitterbasierten Fusion sollte ein Schwerpunkt auf der Auswertung und Präsentation der Gitterdatenstruktur für Applikationen liegen.

A Koordinatensysteme

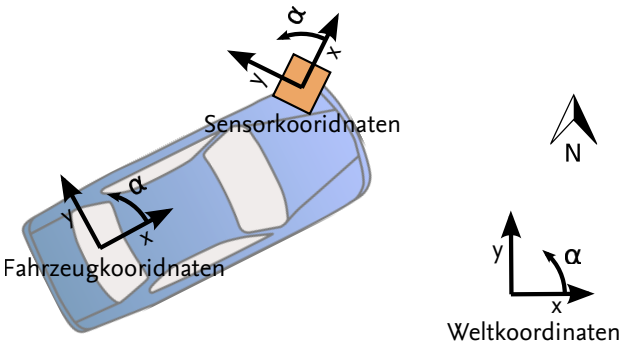


Abbildung A.1: Ursprünge der eingesetzten Koordinatensysteme

Im Rahmen dieser Arbeit werden im Wesentlichen Koordinaten in vier unterschiedlichen Koordinatensystemen eingesetzt. Unterschieden wird dabei zwischen Sensor-, Fahrzeug-, Welt- und Gitterkoordinaten (siehe Abbildung A.1):

Weltkoordinaten: Als Weltkoordinaten wird in dieser Arbeit ein ortsfestes Koordinatensystem betrachtet. Eingesetzt werden im Versuchsträger, je nach Betriebsort, das UTM32- (Hager u. a., 1986) oder das Gauß-Krüger-Koordinatensystem (Kohlstock, 2004, Seite 37f). Beide Koordinatensysteme teilen die Erde in Streifen (3° und 6° Breite) und unterscheiden sich in den eingesetzten geometrischen Grundkörpern zur Verebenung des Erdellipsoids.

Fahrzeugkoordinaten: Unter Fahrzeugkoordinaten wird ein kartesisches Koordinatensystem verstanden, das relativ zum Fahrzeug definiert ist. Im Projekt Stadtpilot wurde ein System nach NORM DIN 70000 (1994) / NORM ISO 8855 (1991) gewählt. Es besitzt seinen Ursprung in der Mitte der Hinterachse. Die X-Achse zeigt zur Fahrzeugfront, während die Y-Achse zur linken Fahrzeugseite zeigt.

Sensorkoordinaten: Der Ursprung dieses Koordinatensystems liegt im Befestigungsort des jeweiligen Sensors und ist relativ zum Fahrzeugkoordinatensystem definiert. Die Orientierung des Koordinatensystems ist sensorspezifisch. Je nach Sensortyp kann hier ein kartesisches oder ein Polarkoordinatensystem zur Anwendung kommen.

Gitterkoordinaten: Gitterkoordinaten beschreiben ein diskretes kartesisches Koordinatensystem. Der Ursprung des Koordinatensystems befindet sich dabei an der linken unteren Ecke einer Gitterdatenstruktur und ist relativ zu einem Ursprungskoordinatensystem (z. B. Weltkoordinaten) definiert. Die Koordinaten erstrecken sich ausschließlich in den positiven Bereich des Koordinatenkreuzes. Die X-Achse des Systems zeigt nach rechts, während die Y-Achse nach oben zeigt.

Transformation zwischen Koordinatensystemen

Die Transformation zwischen Sensor-, Fahrzeug- und Weltkoordinaten kann durch eine oder mehrere Transformationsmatrizen T geschehen. Ist ein Koordinatensystem A beispielsweise um den Winkel α gegenüber einem anderen System B gedreht und um den Vektor $\Delta = [\ x_\Delta \quad y_\Delta\ ]^T$ verschoben, so lautet die Transformationsvorschrift folgendermaßen:

$$\hat{x}_A = [\ x_A \quad y_A\ ]^T \tag{A.1}$$

$$T = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \tag{A.2}$$

$$\hat{x}_B = \hat{x} \cdot T + \Delta \tag{A.3}$$

Um eine Transformation von Gitter- in das zugehörige Ursprungskoordinatensystem durchzuführen, muss zunächst die Diskretisierung ς der Gitterdatenstruktur bekannt sein. Anschließend kann beispielsweise eine Weltkoordinate in eine Gitterkoordinate transformatiert werden bzw. umgekehrt:

$$\hat{x}_{Gitterkoordinaten} = \lfloor \hat{x}_{Weltkoordinaten} \frac{1}{\varsigma} \rfloor \tag{A.4}$$

$$\hat{x}'_{Weltkoordinaten} = \hat{x}_{Gitterkoordinaten} \cdot \varsigma \tag{A.5}$$

Dabei ist zu beachten, dass bei der Transformation zwischen Gitter- und dem Ursprungskoordinaten Diskretisierungsfehler auftreten, sodass hier im Vergleich zwischen den anderen Koordinatensystemen keine symmetrische Abbildung vorliegt.

Fehlerfortpflanzung zwischen Koordinatensystemen

Da Messungen der Sensoren durch Messfehler verfälscht werden, ergänzt sich ein Zustandsvektor zumeist durch die Standardabweichung der einzelnen Messgrößen. Diese Abweichungen müssen bei der Transformation von einem Koordinatensystem in ein anderes mit berücksichtigt werden. Nach dem gaußschen Fehlerfortpflanzungsgesetz (Bronstein u. a., 2001, Seite 816f) lassen sich die Standardabweichungen $\sigma x_A, \sigma y_A$ zu den Koordinaten $[\ x_A \quad y_A\ ]^T$ eines kartesischen Koordinatensystems A in ein anderes Koordinatensystem B berechnen. Die Beziehung zwischen den Koordinatensystemen ist dabei durch eine Translation $\Delta x, \Delta y$ und eine Rotation α sowie die bekannten Standardabweichungen $[\ \sigma x \quad \sigma y \quad \sigma\alpha\ ]^T$ gegeben:

$$f_x(\alpha, x, y) = \sin\alpha \cdot \sqrt{x^2 + y^2} \tag{A.6}$$

$$\sigma x_B^2 = (\frac{\partial f_x}{\partial \alpha} \cdot \sigma\alpha + \frac{\partial f_x}{\partial x} \cdot \sigma x + \frac{\partial f_x}{\partial y} \cdot \sigma y)^2 + \Delta x^2 + \sigma x_A^2 \tag{A.7}$$

$$f_y(\alpha, x, y) = \cos\alpha \cdot \sqrt{x^2 + y^2} \tag{A.8}$$

$$\sigma y_B^2 = (\frac{\partial f_y}{\partial \alpha} \cdot \sigma\alpha + \frac{\partial f_y}{\partial x} \cdot \sigma x + \frac{\partial f_y}{\partial y} \cdot \sigma y)^2 + \Delta y^2 + \sigma y_A^2 \tag{A.9}$$

B Beschreibung des Referenzsystems

Wie der Versuchsträger des Stadtpilot-Projekts ist auch das Referenzsystem auf Basis eines VW Passat Variant aufgebaut. Zur Bestimmung der Position ist es mit einer Trägheitsnavigationsplattform und einem GPS-Empfänger ausgestattet. Als Trägheitsplattform kommt das Gerät iNAV-RQH-100X des Herstellers iMAR zum Einsatz. Es bestimmt die Drehraten des Fahrzeugs mit einer Frequenz von bis zu $300Hz$ und einer Abweichung von weniger als $0,002\frac{deg}{h}$ (iMAR GmbH, 2010, Seite 12). Als GPS-Empfänger wird ein Novatel OEMV (L1/L2 GPS) mit der Antenne Novatel GPS-703-GGG genutzt. Die Daten werden während einer Messfahrt aufgezeichnet und anschließend mit der Software WayPoint Inertial Explorer Version 8.30.2105 miteinander fusioniert (Roesler u. Martell, 2009). Die Postprocessing-Software berechnet nach dem Tightly-Coupled-Verfahren eine exakte Position und Geschwindigkeit des Referenzfahrzeugs sowie die zugehörigen Standardabweichungen. Als Größen stehen die X-, Y- und Z-Position, die Geschwindigkeit, die Bewegungsrichtung sowie deren zugehörige Standardabweichungen des Referenzfahrzeugs zur Verfügung.

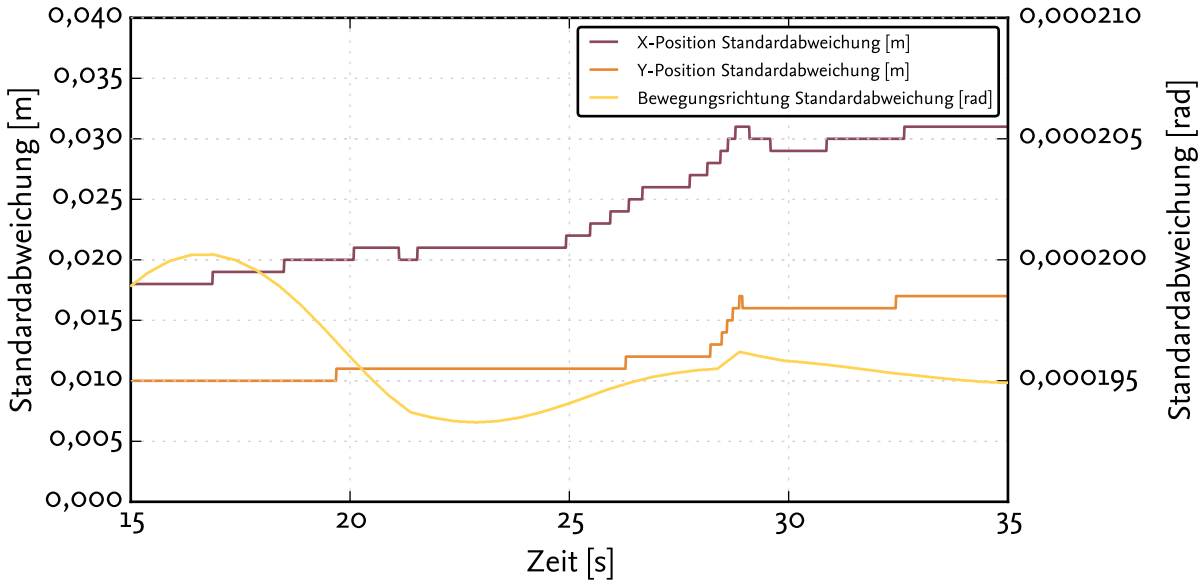


Abbildung B.1: Standardabweichung der X-, Y-Position und der Bewegungsrichtung des Referenzfahrzeugs in Situation 2 aus Abschnitt 16.1.1

Durch das eingesetzte Postprocessing-Verfahren lassen sich sehr genaue Werte für den Zustand des Referenzfahrzeugs bestimmen. Abbildung B.1 zeigt die Standardabweichung der Zustandsgrößen des Referenzfahrzeugs während des Überholmanövers aus Abschnitt 16.1.1. In dieser Situation befindet sich die Standardabweichung der X- und Y-Position des Fahrzeugs im einstelligen Zentimeterbereich und liegt damit deutlich unterhalb der Auflösung der eingesetzten Sensoren.

C Abschätzung der Latenz und von Gütegrößen im Projekt Stadtpilot

Basierend auf einem Workshop am 17. Januar 2012 in den Werkstätten der TU Braunschweig am Campus Nord werden im Folgenden Abschätzungen über die Gesamtlatenz sowie über die Güte zu Positions- und Geschwindigkeitswerten der Sensoren gemacht. Anwesend waren bei dem Workshop die Kollegen Tobias Nothdurft, Falko Saust, Andreas Reschka, Simon Ulbrich und Sebastian Ohl.

C.1 Abschätzung der Gesamtlatenz

Innerhalb des Projekts wurden zwei kritische Pfade identifiziert, die im Weiteren kurz vorgestellt werden. Die angegebenen Latenzen ergeben sich dabei durch Abschätzungen aufgrund der eingesetzten Zykluszeiten der Module oder durch Abschätzung der benötigten Zeit zur Berechnung der Ergebnisse. Als maximale zulässige Zeit des Systems in Notsituationen zwischen Ereignis und Anliegen des Signals am Steuergerät (Bremse, Lenkung) wird dabei eine Sekunde angenommen (Vorbremszeit) (Bürger u. a., 1998, Seite 77ff). Diese setzt sich aus der Zeit zur Gefahrenerkennung ($350ms$ Saccade incl. Fokussierung und $\geq 400ms$ Erkennung), der menschlichen Reaktionszeit (bei visuellen Reizen $220 - 250ms$, bei akustischen Reizen $160 - 190ms$) und der Brems- bzw. Lenkaktivierungszeit (fahrzeugabhängig) zusammen. Somit addiert sich die Gesamtreaktionszeit zu maximal $1000ms$.

Als kritische Pfade wurden einerseits die Reaktion auf ein plötzlich auftretendes Hindernis sowie der Abbruch eines Fahrstreifenwechselmanövers gewählt. Bei der Reaktion auf ein plötzlich auftretendes Hindernis wird lediglich die Längsaktorik angesteuert, während bei dem anderen Pfad lediglich die Queraktorik genutzt wird.

Für die erste Situation wurden folgende Zeiten angenommen:

Modul	Zykluszeit	Latenz
Sensorik	20 Hz	Bis zu $300ms$ (abhängig von der aktuellen Situation)
Kommunikation		$20ms$
PursuiteTargetSeeker		$80ms$
ActiveSegment		$20ms$
Kommunikation		$10ms$
Berechnung Regler		$10ms$
Busansteuerung (nicht Teil der Vorbremszeit)	25 Hz	$40ms$
Bremsensteuergerät (nicht Teil der Vorbremszeit)	25 Hz	$40ms$
Gesamt (nur Vorbremszeit)		$440ms$
Gesamt		$520ms$

Für den zweiten Pfad wurden folgende Zeiten angenommen:

Modul	Zykluszeit	Latenz
Sensorik	20 Hz	Bis zu 300ms (abhängig von der aktuellen Situation)
Kommunikation		20ms
EnvironmentInformationSystem	10 Hz	200ms
DecisionUnit		500ms
Spurplanung		10ms
ActiveSegment		20ms
Kommunikation		10ms
Berechnung Regler		10ms
Busansteuerung (nicht Teil der Vorbremszeit)	25 Hz	40ms
Lenkungssteuergerät (nicht Teil der Vorbremszeit)	25 Hz	40ms
Gesamt (nur Vorbremszeit)		1070ms
Gesamt		1150ms

Zu sehen ist, dass für den Längsaktorik-Pfad die geforderte Zeit erheblich unterschritten wird. Für den Fall des Abbruchs eines Fahrstreifenwechsels wird diese jedoch leicht überschritten. Da dies aber zumeist kein spontanes Manöver, sondern eine länger geplante Entscheidung ist, stellt die Überschreitung hier kein Problem dar.

C.2 Geforderte Güte der Zustandsgrößen der Sensordaten

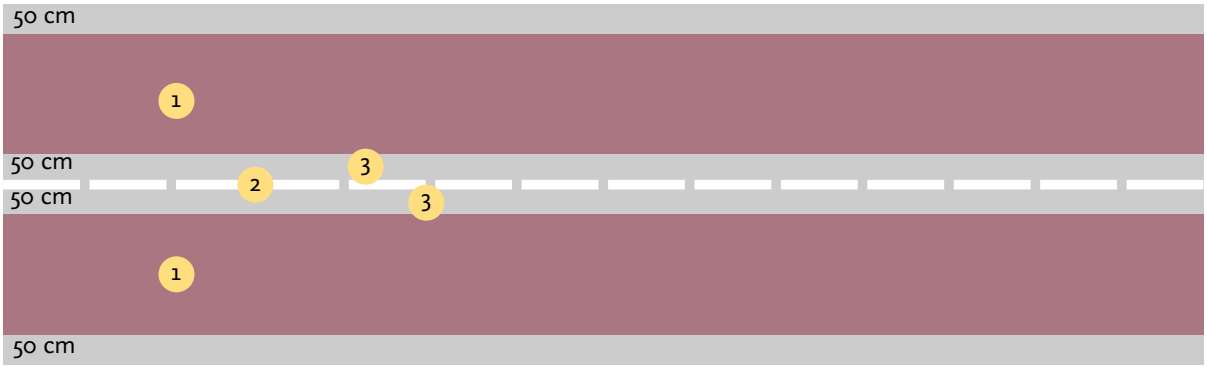


Abbildung C.1: Bereiche der Fahrstreifenzuordnung. Objekthypothesenbreite 50cm, 1: korrekte Zuordnung, 2: keine Zuordnung, 3: Zuordnung am Rand des Fahrstreifens

Festgelegt wurden Grenzen für die Genauigkeit der Geschwindigkeit sowie die Genauigkeit in X- und Y-Richtung.

Bestimmt wurde die Genauigkeit in X- und Y-Richtung zu 50cm. Dieser Wert leitet sich aus den Grenzwerten der Fahrstreifenzuordnung des Umfeldinformationssystems her (siehe Abbildung C.1). Repräsentiert die Objekthypothese lediglich einen Punkt im Raum und liegt mehr als 50cm neben der realen Position, so wird die Objekthypothese entweder keinem

oder eventuell dem falschen Fahrstreifen zugeordnet. Ein sicherer Spurwechsel bzw. eine Folgefahrt ist dann nicht mehr möglich.

Zusätzlich zur Position stellt die Genauigkeit der Breite einer Objekthypothese eine wesentliche Größe dar. Um eine sichere Zuordnung zu einem Fahrstreifen zu ermöglichen, soll sie eine Abweichung von $50cm$ nicht überschreiten. Eine fehlerhafte Zuordnung zu einem Fahrstreifen könnte sonst die Folge sein.

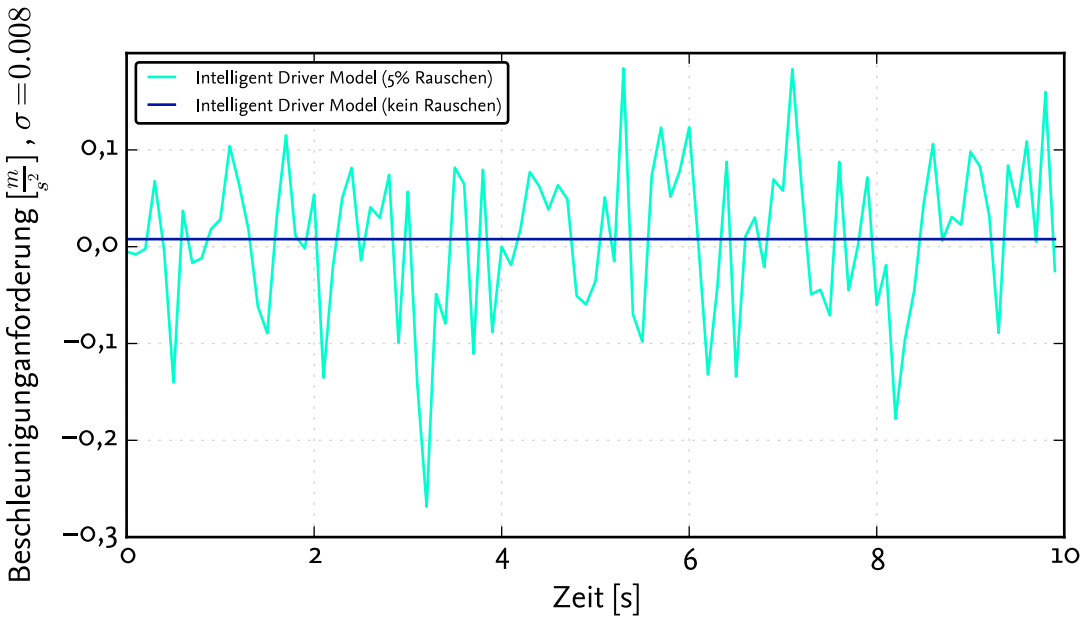


Abbildung C.2: Auswirkung eines fehlerhaften Geschwindigkeitswertes einer Objekthypothese auf die Beschleunigungsanforderung an den Versuchsträger durch das Folgefahrtmodul.

Die Forderung zur Genauigkeit der Geschwindigkeit einer Objekthypothese wird mit ±5% festgelegt, da nach Einschätzung der anwesenden Entwickler dabei eine komfortable Folgefahrt möglich ist. In Abbildung C.2 wird eine Folgefahrtsituation simuliert. Dabei bewegt sich das simulierte Fahrzeug in einem Abstand von $50m$ mit einer Geschwindigkeit von $13,8\frac{m}{s}$ vor dem Versuchsträger. Die Geschwindigkeit des Versuchsträgers beträgt ebenfalls $13,8\frac{m}{s}$. Zur Simulation der Auswirkungen ist der Geschwindigkeitswert der Objekthypothese mit einem weißen gaußschen Rauschen von 5% bzw. $0,69\frac{m}{s}$ behaftet. Das im Folgefahrtmodul eingesetzte Intelligent Driver Model (Treiber u. a., 2000) berechnet aus dieser Objekthypothese eine Beschleunigungsanforderung für den Geschwindigkeitsregler. Die Auswirkungen dieses Fehlers sind in Abbildung C.2 dargestellt. Der Beschleunigungswert würde im rauschfreien Fall $a = 0,008\frac{m}{s^2}$ betragen. Bedingt durch den Fehler schwankt die Anforderung um ca. $\pm 0,2\frac{m}{s^2}$.

Veröffentlichungen

[Basarke u. a. 2008] Basarke, C. ; Berger, C. ; Berger, K. ; Cornelsen, K. ; Doering, M. ; Effertz, J. ; Form, T. ; Gülke, T. ; Graefe, F. ; Hecker, P. ; Homeier, K. ; Klose, F. ; Lipski, C. ; Magnor, M. ; Morgenroth, J. ; Nothdurft, T. ; Ohl, S. ; Rauskolb, F. ; Rumpe, B. ; Schumacher, W. ; Wille, J. ; Wolf, L.: 2007 DARPA Urban Challenge Team CarOLO - Technical Paper / TU Braunschweig. `http://www.sse-tubs.de/publications/CarOLO-TR.pdf`, Abruf: 24.02.2013. 2008. – Forschungsbericht

[Form u. a. 2008] Form, T. ; Effertz, J. ; Ohl, S. ; Wille, J. M.: Urban Challenge 2007, Lessons learned - Team CarOLO. In: Gesamtzentrum für Verkehr Braunschweig e.V. (Hrsg.): *AAET - Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*. Braunschweig, Februar 2008 (AAET)

[Holldorb u. a. 2011] Holldorb, Christian ; Häusler, Katharina ; Maurer, Markus ; Ohl, Sebastian: Autonomes Fahren für den Straßenbetriebsdienst / Hochschule Biberach, Institut für Immobilienökonomie und Projektmanagement (IIP). 2011. – Forschungsbericht. – Ergänzungsbericht zum Forschungsbericht „Informations- und Kommunikationstechniken zur Optimierung des Betriebsdienst-Managements", (Holldorb u. a., 2011b)

[Nothdurft u. a. 2011] Nothdurft, Tobias ; Hecker, Peter ; Ohl, Sebastian ; Saust, Falko ; Maurer, Markus ; Reschka, Andreas ; Böhmer, Jürgen R.: Stadtpilot: First Fully Autonomous Test Drives in Urban Traffic. In: *Proceedings of the 14th International IEEE Annual Conference on Intelligent Transportation Systems*. Washington, DC, USA, Oktober 2011 (ITSC), S. 919–924

[Ohl 2013] Ohl, Sebastian: Static Software Architecture of the Sensor Data Fusion Module of the Stadtpilot Project. In: Maurer, Markus ; Winner, Hermann: *Automotive Systems Engineering*. Berlin Heidelberg : Springer, 2013, S. 81–109

[Ohl u. a. 2012] Ohl, Sebastian ; Häusler, Katharina ; Maurer, Markus ; Holldorb, Christian: Autonomes Fahren im Straßenbetriebsdienst auf Autobahnen. In: Intelligente Transport- und Verkehrssysteme und -dienste Niedersachsen e.V. (Hrsg.): *AAET 2012, Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*. Braunschweig, Februar 2012 (AAET), S. 252–272

[Ohl u. a. 2011] Ohl, Sebastian ; Matthaei, Richard ; Müller, Matthias ; Maurer, Markus: Softwarearchitektur der gitterbasierten Sensordatenfusion des Projekts Stadtpilot. In: Intelligente Transport- und Verkehrssysteme und -dienste Niedersachsen e.V. (Hrsg.): *AAET 2011, Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*. Braunschweig, Februar 2011 (AAET), S. 281–297

[Ohl u. Maurer 2010] Ohl, Sebastian ; Maurer, Markus: Entwicklung einer Software Produktlinie zur flexiblen Erstellung von Umfeldwahrnehmungssystemen. In: Gesamtzentrum für Verkehr Braunschweig e.V. (Hrsg.): *AAET 2010, Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*. Braunschweig, Februar 2010 (AAET), S. 105–120

[Ohl u. Maurer 2011a] Ohl, Sebastian ; Maurer, Markus: A Contour Classifying Kalman Filter Based On Evidence Theory. In: *Proceedings of the 14th International IEEE Annual Conference on Intelligent Transportation Systems*. Washington, DC, USA, Oktober 2011 (ITSC), S. 1392–1397

[Ohl u. Maurer 2011b] Ohl, Sebastian ; Maurer, Markus: Fahrzeugsystemtechnik im Projekt Stadtpilot - am Beispiel der Architektur der objektbasierten Sensordatenfusion. In: *1. Workshop Fahrzeugsystemtechnik*. Wöltingerode, 2011 (FST). – Vortrag

[Ohl u. Maurer 2011c] Ohl, Sebstian ; Maurer, Markus: Ein Kontur schätzendes Kalmanfilter mithilfe der Evidenztheorie. In: Maurer, Markus (Hrsg.) ; Dietmayer, Klaus (Hrsg.) ; Färber, Berthold (Hrsg.) ; Stiller, Christoph (Hrsg.) ; Winner, Hermann (Hrsg.): *7. Workshop Fahrassistenzsysteme*. Walting, April 2011 (FAS), S. 83–94

[Rauskolb u. a. 2009] Rauskolb, Fred ; Berger, Kai ; Lipski, Christian ; Magnor, Marcus ; Cornelsen, Karsten ; Effertz, Jan ; Form, Thomas ; Graefe, Fabian ; Ohl, Sebastian ; Schumacher, Walter ; Wille, Jörn-Marten ; Hecker, Peter ; Nothdurft, Tobias ; Doering, Michael ; Homeier, Kai ; Morgenroth, Johannes ; Wolf, Lars ; Basarke, Christian ; Berger, Christian ; Gülke, Tim ; Klose, Felix ; Rumpe, Bernhard: Caroline: An Autonomously Driving Vehicle for Urban Environments. In: Buehler, Martin (Hrsg.) ; Iagnemma, Karl (Hrsg.) ; Singh, Sanjiv (Hrsg.): *The DARPA Urban Challenge* Bd. 56. Berlin Heidelberg : Springer, 2009, S. 441–508

[Rauskolb u. a. 2008] Rauskolb, Fred W. ; Berger, Kai ; Lipski, Christian ; Magnor, Marcus ; Cornelsen, Karsten ; Effertz, Jan ; Form, Thomas ; Graefe, Fabian ; Ohl, Sebastian ; Schumacher, Walter ; Wille, Jörn-Marten ; Hecker, Peter ; Nothdurft, Tobias ; Doering, Michael ; Homeier, Kai ; Morgenroth, Johannes ; Wolf, Lars ; Basarke, Christian ; Berger, Christian ; Gülke, Tim ; Klose, Felix ; Rumpe, Bernhard: Caroline: An autonomously driving vehicle for urban environments. In: Buehler, Martin (Hrsg.) ; Iagnemma, Karl (Hrsg.) ; Singh, Sanjiv (Hrsg.): *Journal of Field Robotics* Bd. 25. Wiley Periodicals, Inc., August 2008, S. 674–724

[Wille u. a. 2009] Wille, Jörn M. ; Matthaei, Richard ; Ohl, Sebastian ; Saust, Falko ; Maurer, Maurer ; Schumacher, Walter ; Homeier, Kai ; Nothdurft, Tobias ; Sasse, Andreas ; Hecker, Peter ; Wolf, Lars: Der Stadtpilot - Autonomes Fahren auf dem Braunschweiger Stadtring. In: Gesamtzentrum für Verkehr Braunschweig e.V. (Hrsg.): *AAET 2009, Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*. Braunschweig, Februar 2009 (AAET), S. 27–47

Betreute studentische Arbeiten

[Fischer 2013] Fischer, Lukas: *Simulation eines AUTOSAR-Steuergeräts im Automotive Data and Time-Triggered Framework*, Ostfalia Hochschule für angewandte Wissenschaften, Fakultät Informatik, Bachelorarbeit, März 2013

[Gao 2010] Gao, Yang: *Fahrbereichserkennung und -tracking auf Basis von Laserdaten*, Technische Universität Braunschweig, Institut für Regelungstechnik, Studienarbeit, 2010

[Kählke 2010] Kählke, Lars: *Berücksichtigung von redundanten Sichtbereichen in der Trackinitialisierung einer objektbasierten Sensordatenfusion*, Technische Universität Braunschweig, Institut für Regelungstechnik, Bachelorarbeit, November 2010

[Liu 2012] Liu, Shanchun: *Object Detection via a Grid-Based Data Structure using a 2D Split and Merge Algorithm*, Technische Universität Braunschweig, Institut für Regelungstechnik, Studienarbeit, Februar 2012

[Marek 2011] Marek, Gregor: *Entwicklung eines Map-Matching-Verfahrens zur lidarbasierten Positionsstützung*, Technische Universität Braunschweig, Institut für Regelungstechnik, Bachelorarbeit, Mai 2011

[Schütt 2011] Schütt, Thore: *Entwicklung und Umsetzung eines Konzepts zur Markierung von Datenaufzeichnungen im Rahmen des Stadtpilot-Projekts*, Technische Universität Braunschweig, Institut für Regelungstechnik, Bachelorarbeit, April 2011

[Soares 2010] Soares, Bruno Favoreto F.: *Sensor Data Analysis for PreCrash Applications*, Technische Universität Braunschweig, Institut für Regelungstechnik, Diplomarbeit, Februar 2010

[Volz 2012] Volz, Thorsten: *Evolution von Steuergerätesoftware von AUTOSAR Version 2.1 auf Version 3.2 am Beispiel vernetzter Komfortsteuergeräte*, Technische Universität Braunschweig, Institut für Regelungstechnik, Diplomarbeit, Juni 2012

Literaturverzeichnis

[Al-Dhaher u. Mackesy 2004] Al-Dhaher, A.H.G. ; Mackesy, D.: Multi-sensor data fusion architecture. In: *Proceedings of the 3rd IEEE International Workshop on Haptic, Audio and Visual Environments and Their Applications*. Ottawa, Kanada, Oktober 2004 (HAVE), S. 159–163

[Alami u. a. 1998] Alami, R. ; Chatila, R. ; Fleury, S. ; Ghallab, M. ; Ingrand, F.: An Architecture for Autonomy. In: *International Journal of Robotics Research* Bd. 17 (1998), Nr. 4, S. 315–337

[Alexander u. a. 1977] Alexander, Christopher ; Ishikawa, Sara ; Silverstein, Murray: *A Pattern Language: Towns, Buildings, Construction*. New York : Oxford University Press, 1977

[Arkin u. a. 1991] Arkin, E.M. ; Chew, L.P. ; Huttenlocher, D.P. ; Kedem, K. ; Mitchell, J.S.B.: An efficiently computable metric for comparing polygonal shapes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* Bd. 13 (1991), März, Nr. 3, S. 209–216

[Audi Electronic Venture GmbH 2011] Audi Electronic Venture GmbH: *ADTF - Automotive Data and Timeriggered Framework - Benutzerhandbuch*. Sachsstraße 18, 85080 Gaimersheim, 2011

[AutoNOMOUS Labs 2011] AutoNOMOUS Labs: *Autonomous Car Navigates the Streets of Berlin*. Pressemitteilung. `http://autonomos.inf.fu-berlin.de/news/press-release-92011`. September 2011, Abruf: 16.04.2012

[BBC News 2011] BBC News: *Driverless car: Google awarded US patent for technology*. `http://www.bbc.co.uk/news/technology-16197664`. Dezember 2011, Abruf: 17.04.2012

[Bedworth u. O'Brien 1999] Bedworth, M. D. ; O'Brien, J.C.: The Omnibus Model: A New Model for Data Fusion? In: *Proceedings of the 2nd International Conference on Information Fusion*. Sunnyvale, CA, USA, Juli 1999 (Fusion), S. 437–444

[Bertozzi u. a. 2011] Bertozzi, M. ; Bombini, L. ; Broggi, A. ; Buzzoni, M. ; Cardarelli, E. ; Cattani, S. ; Cerri, P. ; Coati, A. ; Debattisti, S. ; Falzoni, A. ; Fedriga, R.I. ; Felisa, M. ; Gatti, L. ; Giacomazzo, A. ; Grisleri, P. ; Laghi, M.C. ; Mazzei, L. ; Medici, P. ; Panciroli, M. ; Porta, P.P. ; Zani, P. ; Versari, P.: VIAC: an Out of Ordinary Experiment. In: *Proceedings of the Intelligent Vehicles Symposium*. Baden-Baden, Juni 2011 (IV), S. 175–180

[Bertsekas 1991] Bertsekas, Dimitri P.: An Auction Algorithm for Shortest Paths. In: *SIAM Journal on Optimization* Bd. 1 (1991), Nr. 4, S. 425–447

[Blackman u. Popoli 1999] BLACKMAN, S. ; POPOLI, R.: *Design and Analysis of Modern Tracking Systems*. Norwood, MA, USA : Artech House Publishers, 1999

[Blom 1984] BLOM, H. A. P.: An Efficient Filter for Abruptly Changing Systems. In: *Proceedings of the 23rd IEEE Conference on Decision and Control*. Las Vegas, NV, USA, Dezember 1984 (CDC), S. 656–658

[BMW Group München 2009] BMW GROUP MÜNCHEN: *Der neue BMW 7er: Entwicklung und Technik*. Wiesbaden : Vieweg+Teubner Verlag, 2009 (ATZ /MTZ-Typenbuch)

[Bombini u. a. 2009] BOMBINI, Luca ; CATTANI, Stefano ; CERRI, Pietro ; FEDRIGA, Rean I. ; FELISA, Mirko ; PORTA, Pier P.: Test-bed for Unified Perception & Decision Architecture. In: *Proceedings of the 13th International Forum on Advanced Microsystems for Automotive Applications*. Berlin, Mai 2009

[Borges u. Aldon 2004] BORGES, Geovany A. ; ALDON, Marie-José: Line Extraction in 2D Range Images for Mobile Robotics. In: *Journal of Intelligent & Robotic Systems*. Springer Netherlands Bd. 40 (2004), Nr. 3, S. 267–297

[Boyd 1976] BOYD, R. R.: *A Discourse on Winning and Losing*. 1976. – Folien, Air University Library, Maxwell, AL, USA

[Bresenham 1965] BRESENHAM, J. E.: Algorithm for computer control of a digital plotter. In: *IBM Systems Journal* Bd. 4 (1965), Nr. 1, S. 25–30

[Brin 2012] BRIN, Sergey: *Bringing self-driving cars to NASCAR*. Google Blog. `http://googleblog.blogspot.de/2012/03/bringing-self-driving-cars-to-nascar.html`. April 2012, Abruf: 17.04.2012

[Bronstein u. a. 2001] BRONSTEIN, I.N. ; SEMEDJAJEW, K.A. ; MUSIOL, G. ; MÜHLIG, H.: *Taschenbuch der Mathematik*. 5. Auflage. Thun und Frankfurt am Main : Verlag Harri Deutsch, 2001

[Brumback u. Srinath 1987] BRUMBACK, B. ; SRINATH, M.: A Chi-square Test for Fault-Detection in Kalman filters. In: *IEEE Transactions on Automatic Control* Bd. 32 (1987), Juni, Nr. 6, S. 552–554

[Bruyninckx 2001] BRUYNINCKX, H.: Open Robot Control Software: the OROCOS project. In: *Proceedings of the IEEE International Conference on Robotics and Automation* Bd. 3. Seoul, Südkorea, Mai 2001 (ICRA), S. 2523–2528

[Bundesanstalt für Straßenwesen 2012] BUNDESANSTALT FÜR STRASSENWESEN: Rechtsfolgen zunehmender Fahrzeugautomatisierung. In: *Berichte der Bundesanstalt für Straßenwesen*. Bergisch Gladbach, Januar 2012 (Fahrzeugtechnik Heft F 83)

[Bürger u. a. 1998] BÜRGER, Heribert ; RAUCHECKER, Franz ; SACHER, Fritz ; WIELKE, Bernhard ; FUCIK, Robert (Hrsg.) ; HARTL, Franz (Hrsg.) ; SCHLOSSER, Horst (Hrsg.) ; WIELKE, Bernhard (Hrsg.): *Handbuch des Verkehrsunfalls*. Bd. 2. Wien, Österreich : MANZ'sche Verlags- und Universitätsbuchhandlung AG, 1998

[Buschmann u. a. 2007] Buschmann, F. ; Henney, K. ; Schmidt, D.C.: *Pattern-oriented Software Architecture - A Pattern Language for Distributed Computing*. Bd. 4. West Sussex, England : John Wiley & Sons Ltd., 2007

[Buschmann u. a. 1996] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern-oriented Software Architecture - A System of Patterns*. Bd. 1. West Sussex, England : John Wiley & Sons Ltd., 1996

[Carvalho u. Heinzelman 2003] Carvalho, Hervaldo S. ; Heinzelman, Wendi B.: A General Data Fusion Architecture. In: *Proceedings of the 6th International Conference on Information Fusion*. Cairns, Australien, Juli 2003 (Fusion), S. 1465–1472

[Clements u. a. 2010] Clements, Paul ; Bachmann, Felix ; Bass, Len ; Garlan, David ; Ivers, James ; Little, Reed ; Nord, Robot ; Stafford, Judith: *Documenting Software Architectures: Views and Beyond*. 2. Auflage. Upper Saddle River, NJ, USA : Addison-Wesley, 2010

[Cox 1964] Cox, H.: On the Estimation of State Variables and Parameters for Noisy Dynamic Systems. In: *IEEE Transactions on Automatic Control* Bd. 9 (1964), Januar, Nr. 1, S. 5–12

[Darms u. Winner 2005] Darms, M. ; Winner, H.: A modular system architecture for sensor data processing of ADAS applications. In: *Proceedings of IEEE Intelligent Vehicles Symposium*. Las Vegas, NV, USA, Juni 2005 (IV), S. 729–734

[Darms 2007] Darms, Michael: *Eine Basis-Systemarchitektur zur Sensordatenfusion von Umfeldsensoren für Fahrerassistenzsysteme*, Technische Universität Darmstadt, Fachgebiet Fahrzeugtechnik, Dissertation, Januar 2007

[DARPA 2003] DARPA: *DARPA Plans Grand Challenge for Robotic Ground Vehicles*. `http://archive.darpa.mil/grandchallenge04/media/announcement.pdf`. 2003, Abruf: 12.04.2012

[DARPA 2005] DARPA: *Grand Challenge 2005: DARPA Schedules Autonomous Robotic Ground Vehicles Event*. `http://archive.darpa.mil/grandchallenge05/InitialPressRelease.pdf`. 2005, Abruf: 12.04.2012

[DARPA 2006a] DARPA: *DARPA Announces Third Grand Challenge Urban Challenge Moves to the City*. `http://archive.darpa.mil/grandchallenge/docs/PR_UC_Announce_Update_12_06.pdf`. 2006, Abruf: 12.04.2012

[DARPA 2006b] DARPA: *Urban Challenge Technical Evaluation Criteria*. `http://archive.darpa.mil/grandchallenge/docs/Technical_Evaluation_Criteria_031607.pdf`. 2006, Abruf: 05.06.2012

[Dasarathy 1997] Dasarathy, B.V.: Sensor Fusion Potential Exploitation-Innovative Architectures and Illustrative Applications. In: *Proceedings of the IEEE* Bd. 85 (1997), Januar, Nr. 1, S. 24–38

[Devore u. Berk 2012] DEVORE, Jay L. ; BERK, Kenneth N.: *Modern Mathematical Statistics with Applications*. 2. Auflage. New York, NY, USA : Springer, 2012 (Springer Texts in Statistics)

[Deza u. Deza 2009] DEZA, Michel M. ; DEZA, Elena: *Encyclopedia of Distances*. Berlin Heidelberg : Springer, 2009

[Dietmayer u. a. 2005] DIETMAYER, Klaus ; KIRCHNER, Alexander ; KÄMPCHEN, Nico: Fusionsarchitekturen zur Umfeldwahrnehmung für zukünftige Fahrerassistenzsysteme. In: MAURER, Markus ; STILLER, Christoph (Hrsg.): *Fahrerassistenzsysteme mit maschineller Wahrnehmung*. Berlin Heidelberg : Springer, 2005, S. 59–88

[Durrant-Whyte u. a. 1990] DURRANT-WHYTE, H.F. ; RAO, B.Y.S. ; HU, H.: Toward a Fully Decentralized Architecture for Multi-Sensor Data Fusion. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 2. Cincinnati, OH, USA, Mai 1990 (ICRA), S. 1331–1336

[Effertz 2008] EFFERTZ, Jan: Sensor Architecture and Data Fusion for Robotic Perception in Urban Environments at the 2007 DARPA Urban Challenge. In: *Proceedings of the 2nd international conference on Robot vision*. Auckland, Neuseeland, Februar 2008 (RobVis), S. 275–290

[Effertz 2009] EFFERTZ, Jan: *Autonome Fahrzeugführung in urbaner Umgebung durch Kombination objekt- und kartenbasierter Umfeldmodelle*, Technische Universität Braunschweig, Institut für Regelungstechnik, Dissertation, Februar 2009

[Elfes 1990] ELFES, A.: Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception. In: *Proceedings of the 6th Conference Annual Conference on Uncertainty in Artificial Intelligence*. Cambridge, MA, USA, Juli 1990, S. 136–146

[Europäisches Parlament 2009] EUROPÄISCHES PARLAMENT, Rat: *Verordnung (EG) Nr. 661/2009 des Europäischen Parlaments und des Rates vom 13. Juli 2009 über die Typgenehmigung von Kraftfahrzeugen, Kraftfahrzeuganhängern und von Systemen, Bauteilen und selbstständigen technischen Einheiten für diese Fahrzeuge hinsichtlich ihrer allgemeinen Sicherheit (Text von Bedeutung für den EWR)*. Verordnung. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2009:200:0001:0024:DE:PDF`. Juli 2009, Abruf: 30.04.2012

[Fassbender 2012] FASSBENDER, Torsten: *Videobasierte Fußgängererkennung für autonome Fahrzeuge*, Freie Universität Berlin, Institut für Informatik, Diplomarbeit, Februar 2012

[Foessel 2000] FOESSEL, Alex: Radar Sensor Model for Three-Dimensional Map Building. In: GAGE, Douglas W. (Hrsg.) ; CHOSET, Howie M. (Hrsg.) ; STEIN, Matthew R.: *Mobile Robots XV and Telemanipulator and Telepresence Technologies VII* Bd. 4195, SPIE, November 2000

[Gad u. Farooq 2002] GAD, A. ; FAROOQ, M.: Data Fusion Architecture for Maritime Surveillance. In: *Proceedings of the 5th International Conference on Information Fusion* Bd. 1. Annapolis, MD, USA, Juli 2002 (Fusion), S. 448–455

[Gamma u. a. 2004] Gamma, Erich ; Helm, Richard ; Johnsen, Ralph ; Vlissides, John: *Entwurfsmuster : Elemente wiederverwendbarer objektorientierter Software*. München : Addison-Wesley, 2004

[Gordon u. a. 1993] Gordon, N.J. ; Salmond, D.J. ; Smith, A.F.M.: Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In: *IEEE Radar and Signal Processing* Bd. 140 (1993), April, Nr. 2, S. 107–113

[Guizzo 2011] Guizzo, Erico: *How Google's Self-Driving Car Works*. IEEE Spectrum. `http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works`. Oktober 2011, Abruf: 17.04.2012

[Hager u. a. 1986] Hager, John W. ; Behensky, James F. ; Drew, Brad W.: The Universal Grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS) / Defence Mapping Agency. 1986 (TM8358.2). – Technische Anleitung

[Hella KGaA Hueck & Co 2006] Hella KGaA Hueck & Co: *Infrared Sensor for Lidar Based Distance Measurement*. Rixbecker Str. 75, 59552 Lippstadt, Mai 2006. – Benutzerhandbuch

[Hella KGaA Hueck & Co 2008] Hella KGaA Hueck & Co: *IDIS Infrared Sensor for Lidar Based Distance Measurement, IDIS Version 2.0 for frontend integration for ACC, ACC Stop&Go, Forward Collision Warning (FCW), Collision Mitigation by Braking (CmbB), PreCrash*. Rixbecker Str. 75, 59552 Lippstadt, Dezember 2008. – Version 4.2, Benutzerhandbuch

[Hruschka u. Starke 2011] Hruschka, Peter ; Starke, Gernot: *arc42 Template*. `http://www.arc42.de/`. 2011, Abruf: 01.12.2011

[Ibeo Automobile Sensor GmbH 2006] Ibeo Automobile Sensor GmbH: *ALASCA User Manual*. Fahrenkrön 125, 22179 Hamburg, März 2006. – Benutzerhandbuch

[Ibeo Automobile Sensor GmbH 2008] Ibeo Automobile Sensor GmbH: *IBEO Lux Betriebsanleitung*. Merkurring 20, 22143 Hamburg, August 2008. – Benutzerhandbuch

[iMAR GmbH 2010] iMAR GmbH: *Hardware ICD1 for iNAV-FJI-001-J/Q iNAV-RQH-100x iNAV-FMS-E-DA iNAV-FCAI-E-DA*. Im Reihersbruch 3, 66386 St. Ingbert, Oktober 2010. – Benutzerhandbuch

[Johnson u. a. 1995] Johnson, Ralph ; Gamma, Eric ; Helm, Richard ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Upper Saddle River, NJ, USA : Addison-Wesley, 1995

[Kalman 1960] Kalman, Rudolph E.: A New Approach to Linear Filtering and Prediction Problems. In: *Journal of Basic Engineering* Bd. 82 (1960), März, Nr. D, S. 35–45

[Kemeny 1990] Kemeny, A.: PROMETHEUS - Design Technics. In: *Proceedings of the International Congress on Transportation Electronics*. Dearborn, MI, USA, Oktober 1990 (ITCE), S. 201–207

[Khaleghi u. a. 2011] KHALEGHI, Bahador ; KHAMIS, Alaa ; KARRAY, Fakhreddine O. ; RAZAVI, Saiedeh N.: Multisensor data fusion: A review of the state-of-the-art. In: *Information Fusion* (2011), August. – Elsevier

[Kohlstock 2004] KOHLSTOCK, Peter: *Kartographie: Eine Einführung*. Paderborn : Ferdinand Schöningh, 2004

[Kruchten 1995] KRUCHTEN, Philippe ..: The 4+1 View Model of architecture. In: *IEEE Software* Bd. 12 (1995), November, Nr. 6, S. 42–50

[Kählke 2010] KÄHLKE, Lars: *Berücksichtigung von redundanten Sichtbereichen in der Trackinitialisierung einer objektbasierten Sensordatenfusion*, Technische Universität Braunschweig, Institut für Regelungstechnik, Bachelorarbeit, November 2010

[Kämpchen 2007] KÄMPCHEN, Nico: *Feature-Level Fusion of Laser Scanner and Video Data for Advanced Driver Assistance Systems*, Universität Ulm, Institut für Mess-, Regel- u. Mikrotechnik, Dissertation, 2007

[Köster u. Frankiewicz 2012] KÖSTER, Frank ; FRANKIEWICZ, Tobias: Die Anwendungsplattform Intelligente Mobilität im Kontext der Entwicklung sicherheitsgerichteter Assistenz und Automation für Straßenfahrzeuge. In: INTELLIGENTE TRANSPORT- UND VERKEHRSSYSTEME UND -DIENSTE NIEDERSACHSEN E.V. (Hrsg.): *AAET 2012, Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*. Braunschweig, Februar 2012 (AAET), S. 58–76

[Lalanda 1998] LALANDA, Philippe: Shared repository pattern. In: *Proceedings of the 1998 Pattern Languages of Programs Conference*. Monticello, IL, USA, August 1998 (PLOP), S. 1–10

[Lerro u. Bar-Shalom 1993] LERRO, D. ; BAR-SHALOM, Y.: Tracking With Debiased Consistent Converted Measurements Versus EKF. In: *IEEE Transactions on Aerospace and Electronic Systems* Bd. 29 (1993), Juli, Nr. 3, S. 1015–1022

[Levinson u. a. 2011a] LEVINSON, J. ; ASKELAND, J. ; BECKER, J. ; DOLSON, J. ; HELD, D. ; KAMMEL, S. ; KOLTER, J.Z. ; LANGER, D. ; PINK, O. ; PRATT, V. ; SOKOLSKY, M. ; STANEK, G. ; STAVENS, D. ; TEICHMAN, A. ; WERLING, M. ; THRUN, S.: Towards Fully Autonomous Driving: Systems and Algorithms. In: *Proceedings of the Intelligent Vehicles Symposium*. Baden-Baden, Juni 2011 (IV), S. 163–168

[Levinson u. a. 2011b] LEVINSON, J. ; ASKELAND, J. ; DOLSON, J. ; THRUN, S.: Traffic Light Mapping, Localization, and State Detection for Autonomous Vehicles. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Shanghai, China, Mai 2011 (ICRA), S. 5784–5791

[Levinson u. Thrun 2010] LEVINSON, J. ; THRUN, S.: Robust Vehicle Localization in Urban Environments Using Probabilistic Maps. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Anchorage, AK, USA, Mai 2010 (ICRA), S. 4372–4378

[Li u. Bar-Shalom 1992] Li, X. R. ; Bar-Shalom, Yaakov: Mode-Set Adaptation in Multiple-Model Estimators for Hybrid Systems. In: *Proceedings of the American Control Conference*. Chicago, IL, USA, Juni 1992, S. 1794–1799

[Liu 2012] Liu, Shanchun: *Object Detection via a Grid-Based Data Structure using a 2D Split and Merge Algorithm*, Technische Universität Braunschweig, Institut für Regelungstechnik, Studienarbeit, Februar 2012

[Llinas u. a. 2004] Llinas, James ; Bowman, Christopher ; Rogova, Galina ; Steinberg, Alan ; Waltz, Ed ; White, Frank: Revisiting the JDL Data Fusion Model II. In: *Proceedings of the 7th International Conference on Information Fusion*. Mountain View, CA, USA, Juni 2004 (Fusion), S. 1218–1230

[Lunze 2010] Lunze, Jan: *Regelungstechnik*. Bd. 2. Berlin Heidelberg : Springer, 2010

[Mahalanobis 1936] Mahalanobis, Prasanta C.: On the Generalised Distance in Statistics. In: *Proceedings National Institute of Science, Indien* Bd. 2 (1936), April, Nr. 1, S. 49–55

[Markin u. a. 1997] Markin, M. ; Harris, C. ; Bernhardt, M. ; Austin, J. ; Bedworth, M. ; Greenway, P. ; Johntson, R. ; Little, A. ; Lowe, D.: *Technology Foresight on Data Fusion and Data Processing*. London, England : The Royal Aeronautical Society, 1997

[Matthaei u. a. 2011] Matthaei, Richard ; Dyckmanns, Helgo ; Lichte, Bernd ; Maurer, Markus: Motion Classification for Cross Traffic in Urban Environments Using Laser and Radar. In: *Proceedings of the 14th International Conference on Information Fusion*. Chicago, IL, USA, Juli 2011 (Fusion), S. 1–8

[Maurer 2000] Maurer, Markus: *Flexible Automatisierung von Straßenfahrzeugen mit Rechnersehen*, Universität der Bundeswehr München, Institut für Systemdynamik und Flugmechanik, Dissertation, März 2000

[Maurer u. a. 1996] Maurer, Markus ; Behringer, Reinhold ; Furst, S. ; Thomanek, F. ; Dickmanns, Ernst D.: A Compact Vision System for Road Vehicle Guidance. In: *Proceedings of the 13th International Conference on Pattern Recognition* Bd. 3. Wien, Österreich, August 1996, S. 313–317

[Munkres 1957] Munkres, James: Algorithms for the Assignment and Transportation Problems. In: *Journal of the Society of Industrial and Applied Mathematics* Bd. 5 (1957), März, Nr. 1, S. 32–38

[Munz 2011] Munz, Michael: *Generisches Sensorfusionsframework zur gleichzeitigen Zustands- und Existenzschätzung für die Fahrzeugumfelderkennung*, Universität Ulm, Institut für Mess-, Regel- u. Mikrotechnik, Dissertation, Juli 2011

[Mählisch 2009] Mählisch, Mirko: *Filtersynthese zur simultanen Minimierung von Existenz-, Assoziations- und Zustandsunsicherheiten in der Fahrzeugumfelderfassung mit heterogenen Sensordaten*, Universität Ulm, Institut für Mess-, Regel- u. Mikrotechnik, Dissertation, 2009

[Müller 2012] MÜLLER, Matthias: *Entwicklung einer modularen gridbasierten Umfeldkarte*, Technische Universität Braunschweig, Institut für Regelungstechnik, Studienarbeit, 2012. – Unveröffentlicht

[Nan 2011] NAN, Hao: *Car takes long drive - by itself*. China Daily. `http://www.chinadaily.com.cn/cndy/2011-08/03/content_13037633.htm`. August 2011, Abruf: 17.04.2012

[Neue Zürcher Zeitung 2010] NEUE ZÜRCHER ZEITUNG: *Leonie auf Jungfernfahrt*. `http://www.nzz.ch/nachrichten/panorama/leonie_auf_jungfernfahrt_1.7909243.html`. Oktober 2010, Abruf: 26.4.2012

[Nevada Department of Moto Vehicles 2012] NEVADA DEPARTMENT OF MOTO VEHICLES: *Nevada DMV Issues First Autonomous Vehicle Testing License to Google*. `http://www.dmvnv.com/news/12005-autonomous-vehicle-licensed.htm`. Mai 2012, Abruf: 08.05.2012

[Nguyen u. a. 2007] NGUYEN, Viet ; GÄCHTER, Stefan ; MARTINELLI, Agostino ; TOMATIS, Nicola ; SIEGWART, Roland: A comparison of line extraction algorithms using 2D range data for indoor mobile robotics. In: *Autonomous Robots* Bd. 23 (2007), Nr. 2, S. 97–111

[NORM DIN 70000 1994] NORM DIN 70000: *Straßenfahrzeuge; Fahrzeugdynamik und Fahrverhalten; Begriffe*, Deutsches Institut für Normung e. V., NA 052-01-09 AA - Fahrzeugdynamik und Fahrverhalten, Januar 1994

[NORM FIPS 186-2 2000] NORM FIPS PUB 186-2: *Digital Signature Standard (DSS)*, U.S. Department of Commerce, National Institute of Standards and Technology, Januar 2000

[NORM IEEE Std 1471 2000] NORM IEEE Std. 1471:2000: *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Standards Association, C/S2ESC - Software & Systems Engineering Standards Committee, September 2000

[NORM IEEE Std 610.12 1990] NORM IEEE Std. 610.12:1990: *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Standards Association, C/S2ESC - Software & Systems Engineering Standards Committee, September 1990

[NORM IEEE Std 802.3 2008] NORM IEEE Std. 802.3:2008: *IEEE Standard for Information technology— Telecommunications and information exchange between systems— Local and metropolitan area networks— Specific requirements Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE Standards Association, 802.3 Ethernet Working Group, Dezember 2008

[NORM ISO 8855 1991] NORM ISO Std. 8855:1991: *Road vehicles – Vehicle dynamics and road-holding ability – Vocabulary*, International Organization for Standardization, TC 22/SC 9 - Vehicle dynamics and road-holding ability, Dezember 1991

[Nothdurft u. a. 2011a] NOTHDURFT, Tobias ; HECKER, Peter ; FRANKIEWICZ, Tobias ; GAČNIK, Jan ; KÖSTER, Frank: Reliable Information Aggregation and Exchange for Autonomous Vehicles. In: *Proceedings of the 74th IEEE Vehicular Technology Conference*. San Francisco, CA, USA, September 2011 (VTC)

[Nothdurft u. a. 2011b] Nothdurft, Tobias ; Hecker, Peter ; Ohl, Sebastian ; Saust, Falko ; Maurer, Markus ; Reschka, Andreas ; Böhmer, Jürgen R.: Stadtpilot: First Fully Autonomous Test Drives in Urban Traffic. In: *Proceedings of the 14th International IEEE Annual Conference on Intelligent Transportation Systems*. Washington, DC, USA, Oktober 2011 (ITSC), S. 919–924

[Object Management Group 2005] Object Management Group: *Unified Modeling Language Specification 2.0: Superstructure*, August 2005. – OMG doc. formal/05-07-04

[Ohl 2007] Ohl, Sebastian: *Entwicklung einer Multi-Sensor-Datenfusion für ein autonomes Straßenfahrzeug*, Technische Universität Braunschweig, Institut für Software Systems Engineering, Diplomarbeit, 2007

[Ohl u. a. 2011] Ohl, Sebastian ; Matthaei, Richard ; Müller, Matthias ; Maurer, Markus: Softwarearchitektur der gitterbasierten Sensordatenfusion des Projekts Stadtpilot. In: Intelligente Transport- und Verkehrssysteme und -dienste Niedersachsen e.V. (Hrsg.): *AAET 2011, Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*. Braunschweig, Februar 2011 (AAET), S. 281–297

[Ohl u. Maurer 2011a] Ohl, Sebastian ; Maurer, Markus: A Contour Classifying Kalman Filter Based On Evidence Theory. In: *Proceedings of the 14th International IEEE Annual Conference on Intelligent Transportation Systems*. Washington, DC, USA, Oktober 2011 (ITSC), S. 1392–1397

[Ohl u. Maurer 2011b] Ohl, Sebastian ; Maurer, Markus: Fahrzeugsystemtechnik im Projekt Stadtpilot - am Beispiel der Architektur der objektbasierten Sensordatenfusion. In: *1. Workshop Fahrzeugsystemtechnik*. Wöltingerode, 2011 (FST). – Vortrag

[Pavlidis u. Horowitz 1974] Pavlidis, Theodosios ; Horowitz, Steven.L.: Segmentation of Plane Curves. In: *IEEE Transactions on Computers* Bd. C-23 (1974), August, Nr. 8, S. 860–870

[Pohl u. Rupp 2011] Pohl, Klaus ; Rupp, Chris: *Basiswissen Requirements Engineering: Aus- und Weiterbildung zum Certified Professional for Requirements Engineering ; Foundation-Level nach IREB-Standard*. 3. Auflage. Heidelberg : Dpunkt-Verlag, 2011

[Polychronopoulos u. Amditis 2006] Polychronopoulos, Aristomenis ; Amditis, Angelos: Revisiting JDL model for automotive safety applications: the PF2 functional model. In: *Proceedings of the 9th International Conference on Information Fusion*. Florence, Italien, Juli 2006 (Fusion), S. 1–7

[Pomerleau u. Jochem 1996] Pomerleau, Dean ; Jochem, Todd: Rapidly Adapting Machine Vision for Automated Vehicle Steering. In: *IEEE Expert* Bd. 11 (1996), April, Nr. 2, S. 19–27

[PRORETA 2002] *PRORETA 1 - PRORETA*. `http://www.proreta.tu-darmstadt.de/proreta_1/projekt_proreta_1/index.de.jsp`. 2002-2006, Abruf: 26.04.2012

[Pudenz 2010] PUDENZ, Katrin: *Forschungsfahrzeug Leonie fährt automatisch im realen Stadtverkehr*. ATZ Online. http://www.atzonline.de/Aktuell/Nachrichten/1/12572/Forschungsfahrzeug-Leonie-faehrt-automatisch-im-realen-Stadtverkehr.html. Oktober 2010, Abruf: 26.04.2012

[Rauskolb u. a. 2008] RAUSKOLB, Fred W. ; BERGER, Kai ; LIPSKI, Christian ; MAGNOR, Marcus ; CORNELSEN, Karsten ; EFFERTZ, Jan ; FORM, Thomas ; GRAEFE, Fabian ; OHL, Sebastian ; SCHUMACHER, Walter ; WILLE, Jörn-Marten ; HECKER, Peter ; NOTHDURFT, Tobias ; DOERING, Michael ; HOMEIER, Kai ; MORGENROTH, Johannes ; WOLF, Lars ; BASARKE, Christian ; BERGER, Christian ; GÜLKE, Tim ; KLOSE, Felix ; RUMPE, Bernhard: Caroline: An autonomously driving vehicle for urban environments. In: BUEHLER, Martin (Hrsg.) ; IAGNEMMA, Karl (Hrsg.) ; SINGH, Sanjiv (Hrsg.): *Journal of Field Robotics* Bd. 25. Wiley Periodicals, Inc., August 2008, S. 674–724

[Real-Time Innovations Inc. 2011] REAL-TIME INNOVATIONS INC.: *RTI Data Distribution Service - User's Manual*. 232 E. Java Drive, Sunnyvale, CA 94089, USA, 2011. – Benutzerhandbuch

[Reif 2012] REIF, Konrad: Fahrerassistenzsysteme. In: REIF, Konrad (Hrsg.): *Automobilelektronik*. 4. Auflage. Wiesbaden : Vieweg+Teubner Verlag, 2012, S. 321–367

[Reuschenbach 2011] REUSCHENBACH, Arturo: *iDriver - Eine iPad-Steuerung für ein autonomes Fahrzeug*, Freie Universität Berlin, Institut für Informatik, Masterarbeit, Juni 2011

[Reuschenbach u. a. 2011] REUSCHENBACH, Arturo ; WANG, Miao ; GANJINEH, Tinosch ; GOHRING, Daniel: iDriver - Human Machine Interface for Autonomous Cars. In: *Proceedings of the 8th International Conference on Information Technology: New Generations*. Las Vegas, NV, USA, April 2011 (ITNG), S. 435–440

[Ribo u. Pinz 2001] RIBO, Miguel ; PINZ, Axel: A comparison of three uncertainty calculi for building sonar-based occupancy grids. In: *Robotics and Autonomous Systems* Bd. 35 (2001), Nr. 3–4, S. 201–209

[Roesler u. Martell 2009] ROESLER, Greg ; MARTELL, Hugh: Tightly Coupled Processing of Precise Point Positioning (PPP) and INS Data. In: *Proceedings of the 22nd International Technical Meeting of The Satellite Division of the Institute of Navigation*, 2009 (ION), S. 1898–1905

[Rojo u. a. 2007] ROJO, Javier ; ROJAS, Raul ; GUNNARSSON, Ketill ; SIMON, Mark ; WIESEL, Fabian ; RUFF, Fabian ; WOLTER, Lars ; ZILLY, Frederick ; SANTRAC, Neven ; GENJINEH, Tinosch ; SARKOHI, Arash ; ULBRICH, Fritz ; LATOTZKY, David ; JANKOVIC, Benjamin ; HOHL, Gretta ; WISPEINTNER, Thomas ; MAY, Stefan ; PERVÖLZ, Kai ; NOWAK, Walter ; MAURELLI, Francesco ; DRÖSCHEL, David: Sprit of Berlin: An Autonomous Car for the DARPA Urban Challenge - Hardware and Software Architecture / Freie Universität Berlin. http://archive.darpa.mil/grandchallenge/TechPapers/Team_Berlin.pdf, Abruf: 16.04.2012. 2007. – Forschungsbericht

[Rupp u. a. 2005] Rupp, Chris ; Hahn, Jürgen ; Queins, Stefan ; Jeckle, Mario ; Zengler, Barbara: *UML 2 glasklar: Praxiswissen für die UML-Modellierung*. 2. Auflage. München Wien : Carl Hanser Verlag, 2005

[Saust u. a. 2010] Saust, Falko ; Bley, Oliver ; Kutzner, Ralf ; Wille, Jörn M. ; Friedrich, Bernhard ; Maurer, Markus: Exploitability of vehicle related sensor data in cooperative systems. In: *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems*. Madeira, Portugal, September 2010 (ITSC), S. 1724 –1729

[Schatten u. a. 2010] Schatten, Alexander ; Biffl, Stefan ; Demolsky, Markus ; Gostischa-Franta, Erik ; Östreicher, Thomas ; Winkler, Dietmar: Software-Architektur. In: *Best Practice Software-Engineering*. 1. Auflage. Heidelberg : Spektrum Akademischer Verlag, 2010, S. 199–227

[Scheunert u. a. 2006] Scheunert, Ullrich ; Lindner, Philipp ; Cramer, Heiko ; Tatschke, Thomas ; Polychronopoulos, Aristomenis ; Wanielik, Gerd: Multi Level Processing Methodology for Automotive Applications. In: *Proceedings of the IEEE Intelligent Transportation Systems Conference*. Toronto, Kanada, September 2006 (ITSC), S. 1322–1327

[Schneider 2006] Schneider, Ulrich: *Sensordatenfusion und Fehlerkalibrierung von umfelderkennenden Sensoren eines Straßenfahrzeuges*, Technische Universität Braunschweig, Institut für Regelungstechnik, Dissertation, 2006

[Schumny u. Ohl 1994] Schumny, Harald ; Ohl, Rainer: *Handbuch digitaler Schnittstellen*. 1. Auflage. Braunschweig/Wiesbaden : Vieweg, 1994

[Schutz US 8,078,349] Schutz US 8,078,349 (März 2011). Prada, Gomez ; Luis, Ricardo ; Fairfield, Nathaniel ; Szybalski, Andy ; Nemec, Philip ; Christopher, Urmson; (Erfinder); Google Inc. (Anmelder).

[Schwabe 2012] Schwabe, Manuel: *Auffinden und Verfolgen von Verkehrsteilnehmern für autonome Fahrzeuge mittels Radar*, Freie Universität Berlin, Institut für Informatik, Diplomarbeit, Januar 2012

[Schütt 2011] Schütt, Thore: *Entwicklung und Umsetzung eines Konzepts zur Markierung von Datenaufzeichnungen im Rahmen des Stadtpilot-Projekts*, Technische Universität Braunschweig, Institut für Regelungstechnik, Bachelorarbeit, April 2011

[Shafer 1976] Shafer, Glenn: *A Mathematical Theory of Evidence*. Princeton, NJ, USA : Princeton University Press, 1976

[smart microwave sensors GmbH 2006] smart microwave sensors GmbH: *Universal Medium Range Radar Documentation*. Mittelweg 7, 38106 Braunschweig, Oktober 2006. – Benutzerhandbuch

[smart microwave sensors GmbH 2010] smart microwave sensors GmbH: *Universal Medium Range Radar Documentation*. Mittelweg 7, 38106 Braunschweig, Februar 2010. – Benutzerhandbuch

[Software Engineering Institute at Carnegie Mellon University] SOFTWARE ENGINEERING INSTITUTE AT CARNEGIE MELLON UNIVERSITY: *Community Software Architecture Definitions*. http://www.sei.cmu.edu/architecture/start/community.cfm, Abruf: 20.10.2011

[Sommerville 2004] SOMMERVILLE, Ian: *Software Engineering*. 7. Auflage. Boston : Addison-Wesley, 2004

[Starke u. Hruschka 2011] STARKE, Gernot ; HRUSCHKA, Peter: *Software-Architektur kompakt*. 2. Auflage. Heidelberg : Spektrum Akademischer Verlag, 2011

[Statistisches Bundesamt 2012] STATISTISCHES BUNDESAMT: Verkehr aktuell - Stand 20.04.2012. 2012 (Fachserie 8 Reihe 1.1)

[Steinberg u. a. 1999] STEINBERG, Alan N. ; BOWMAN, Christopher L. ; WHITE, Franklin E.: Revisions to the JDL data fusion model. In: *Sensor Fusion: Architectures, Algorithms, and Applications III*. SPIE Bd. 3719 (1999), Nr. 1, S. 430–441

[Tatschke u. a. 2006] TATSCHKE, Thomas ; PARK, Su-Birm ; AMDITIS, Angelos ; POLYCHRONOPOULOS, Aristomenis ; SCHEUNERT, Ullrich ; AYCARD, Olivier: ProFusion2 — towards a Modular, Robust and Reliable Fusion Architecture for Automotive Environment Perception. In: VALLDORF, Jürgen (Hrsg.) ; GESSNER, Wolfgang (Hrsg.): *Advanced Microsystems for Automotive Applications*. Berlin Heidelberg : Springer, 2006, S. 451–469

[Technische Universität Braunschweig 2010] TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG: *Weltweit erstes automatisches Fahren im realen Stadtverkehr*. https://www.tu-braunschweig.de/presse/medien/presseinformationen?year=2010&pinr=133. Oktober 2010, Abruf: 25.04.2012

[Teichman u. a. 2011] TEICHMAN, Alex ; LEVINSON, Jesse ; THRUN, Sebastian: Towards 3D Object Recognition via Classification of Arbitrary Object Tracks. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Shanghai, China, Mai 2011 (ICRA), S. 4034–4041

[Thompson 1977] THOMPSON, Alan M.: The navigation system of the JPL robot. In: *Proceedings of the 5th international joint conference on Artificial intelligence* Bd. 2. Cambridge, MA, USA, August 1977 (IJCAI), S. 749–757

[Thrun 2010] THRUN, Sebastian: *What we're driving at*. Google Official Blog. http://googleblog.blogspot.de/2010/10/what-were-driving-at.html. Oktober 2010, Abruf: 17.04.2012

[Thrun u. a. 2005] THRUN, Sebastian ; BURGARD, Wolfram ; FOX, Dieter: *Probabilistic robotics*. Cambridge, MA, USA : MIT Press, 2005

[TNO 2009] TNO: *Grand Cooperative Driving Challange accelerates the introduction of cooperative driving*. http://www.gcdc.net/mainmenu/Press/Press_releases/Press_invitationCooperative_Driving. Mai 2009, Abruf: 17.04.2012

[Treiber u. a. 2000] Treiber, Martin ; Hennecke, Ansgar ; Helbing, Dirk: Congested traffic states in empirical observations and microscopic simulations. In: *Physical Review E*. American Physical Society Bd. 62 (2000), August, S. 1805–1824

[Tsugawa u. a. 1979] Tsugawa, Sadayuki ; Yatabe, Teruo ; Hirose, Takeshi ; Matsumoto, Shuntetsu: An automobile with artificial intelligence. In: *Proceedings of the 6th international joint conference on Artificial intelligence* Bd. 2. Tokyo, Japan, 1979 (IJCAI), S. 893–895

[Ulbrich 2011] Ulbrich, Simon: *Intelligent Decision Making and Maneuver Planning for Autonomous Driving in Urban Traffic Environments*, Technische Universität Braunschweig, Institut für Regelungstechnik, Diplomarbeit, Oktober 2011

[Velodyne Lidar, Inc 2008] Velodyne Lidar, Inc: *HDL-64ES2 User's Manual*. 345 Digital Drive, Morgan Hill, CA 95037, USA, Juni 2008. – Benutzerhandbuch

[Vogel u. a. 2008] Vogel, Oliver ; Arnold, Ingo ; Chughtai, Arif ; Ihler, Edmund ; Kehrer, Timo ; Mehlig, Uwe ; Zdun, Uwe: *Software-Architektur: Grundlagen - Konzepte - Praxis*. 2. Auflage. Heidelberg : Spektrum Akademischer Verlag, 2008

[Waibel 2011] Waibel, Markus: *BrainDriver: A Mind Controlled Car*. IEEE Spectrum. `http://spectrum.ieee.org/automaton/robotics/robotics-software/braindriver-a-mind-controlled-car`. Februar 2011, Abruf: 17.04.2012

[Wang u. a. 2011] Wang, Miao ; Ganjineh, Tinosch ; Rojas, Raúl: Action Annotated Trajectory Generation for Autonomous Maneuvers on Structured Road Networks. In: *Proceedings of the 5th International Conference on Automation, Robotics and Applications*. Wellington, Neuseeland, Dezember 2011 (ICARA), S. 67–72

[Watanabe u. a. 1995] Watanabe, Takeo ; Kishimoto, N ; Hayafune, K ; Yamada, K ; Maede, N: Development of an Intelligent Cruise Control System. In: *Proceedings of the 2nd ITS World Congress* Bd. 3. Yokohama, Japan, November 1995 (WC), S. 1229–1235

[Weiss u. a. 2007] Weiss, Thorsten ; Schiele, Bruno ; Dietmayer, Klaus: Robust Driving Path Detection in Urban and Highway Scenarios Using a Laser Scanner and Online Occupancy Grids. In: *Proceedings of the IEEE Intelligent Vehicles Symposium*. Istanbul, Türkei, Juni 2007 (IV), S. 184–189

[Weiss 2011] Weiss, Thorsten-Tobias: *Hochgenaue Positionierung und Kartographie mit Laserscannern für Fahrerassistenzsysteme*, Universität Ulm, Dissertation, Juni 2011

[Wender 2007] Wender, Stefan: *Multisensorsystem zur erweiterten Fahrzeugerfassung*, Universität Ulm, Dissertation, 2007

[White 1988] White, Franklin E.: A Model for Data Fusion. In: *Proceedings of the 1st National Symposium on Sensor Fusion* Bd. 2. Orlando, FL, USA, April 1988

[Wille u. a. 2010] Wille, J.M. ; Saust, F. ; Maurer, M.: Comprehensive Treated Sections in a Trajectory Planner for Realizing Autonomous Driving in Braunschweig's Urban Traffic. In: *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems*. Madeira, Portugal, September 2010 (ITSC), S. 647–652

[Wille u. a. 2009] WILLE, Jörn M. ; MATTHAEI, Richard ; OHL, Sebastian ; SAUST, Falko ; MAURER, Maurer ; SCHUMACHER, Walter ; HOMEIER, Kai ; NOTHDURFT, Tobias ; SASSE, Andreas ; HECKER, Peter ; WOLF, Lars: Der Stadtpilot - Autonomes Fahren auf dem Braunschweiger Stadtring. In: GESAMTZENTRUM FÜR VERKEHR BRAUNSCHWEIG E.V. (Hrsg.): *AAET 2009, Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*. Braunschweig, Februar 2009 (AAET), S. 27–47

[Winner u. a. 2009] WINNER, Hermann ; HAKULI, Stephan ; WOLF, Gabriele (Hrsg.): *Handbuch Fahrerassistenzsysteme Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Wiesbaden : Vieweg+Teubner, 2009

[Wu u. a. 2002] WU, Huadong ; SIEGEL, Mel ; STIEFELHAGEN, Rainer ; YANG, Jie: Sensor Fusion Using Dempster-Shafer Theory. In: *Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference* Bd. 1. Anchorage, AK, USA, August 2002 (IMTC), S. 7–12

[Zhang 1997] ZHANG, Zhengyou: Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting. In: *Image and Vision Computing* Bd. 15 (1997), Nr. 1, S. 59–76